Towards better quality and reliability in the software reuse library environment

Warburton, Kenneth M.

Monterey, California. Naval Postgraduate School

Approved for public release; distribution is unlimited.

TOWARDS BETTER QUALITY AND RELIABILITY IN THE SOFTWARE REUSE LIBRARY ENVIRONMENT

by

Kenneth M. Warburton
Captain, United States Marine Corps
B.S., Auburn University, 1988

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March 1994

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 24 March 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE TOWARDS BETTER QUALITY AND RELIABILITY IN THE SOFTWARE REUSE LIBRARY ENVIRONMENT

5. FUNDING NUMBERS

6. AUTHOR(S) Kenneth M. Warburton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
*A

13. ABSTRACT *(maximum 200 words)*

In today's DoD software environment, where systems of enormous size, complexity and cost are the norm, economic conditions are driving DoD system developers to seek ways to increase productivity while decreasing product defects. To achieve its goals, DoD has taken the approach of integrating reuse into the software development process. In 1992, DISA established its Software Reuse Program to serve as a prototype for the DoD-wide reuse initiative. This thesis will look at DISA's effort to support DoD's reuse vision. Specifically, it will discuss DISA's software reuse library management and will introduce a methodology for the collection and analysis of metrics relating to software performance in order to improve library software quality. This thesis concludes that metrics can play a key role in any organization's software quality program. While metrics alone are not a solution to the reuse quality problem, they are a tool to be used prudently by the software quality manager to manage and improve the quality of organizational software.

14. SUBJECT TERMS Reuse, Reliability, Quality, Metrics			15. NUMBER OF PAGES 104 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)

ABSTRACT

In today's DoD software environment, where systems of enormous size, complexity and cost are the norm, economic conditions are driving DoD system developers to seek ways to increase productivity while decreasing product defects. To achieve its goals, DoD has taken the approach of integrating reuse into the software development process. In 1992, DISA established its Software Reuse Program to serve as a prototype for the DoD-wide reuse initiative. This thesis will look at DISA's effort to support DoD's reuse vision. Specifically, it will discuss DISA's software reuse library management and will introduce a methodology for the collection and analysis of metrics relating to software performance in order to improve library software quality. This thesis concludes that metrics can play a key role in any organization's software quality program. While metrics alone are not a solution to the reuse quality problem, they are a tool to be used prudently by the software quality manager to manage and improve the quality of organizational software.

TABLE OF CONTENTS

I. THE SOFTWARE DILEMMA

A. DOD AND INDUSTRY OVERVIEW

From most accounts, the software industry has been experiencing a "software crisis" since the late 1960s. David Fisher (Institute for Defense Analysis Report P-1191, 1976) offers the following characteristics of the crisis:

- Software often fails leading to poor reliability.
- Software development costs are unpredictable.
- Software is delivered late, often quasi-functional.
- Software is seldom portable among domains.

Because software often exhibits the characteristics described above, industry reportedly spends anywhere from 40% to 70% of its computing budget on software maintenance (Booch, 1987). As a result, in the field of software engineering, efforts to improve software quality and reliability have become the focus of both the government and industry alike.

In today's DoD software environment, where systems of enormous size, complexity and cost are the norm, economic conditions are driving DoD system developers to seek ways to increase productivity while decreasing product defects. DoD can no longer afford to invest substantial sums of money fielding complex weapons or MIS systems that fail to meet

operational expectations due to poor design and reliability. To counter the increasing problem of high cost and poor performance software systems, DoD and its civilian contractors have launched a concerted effort to improve the quality of software by better managing its design and maintenance process.

B. DOD'S RESPONSE TO THE CRISIS

On July 15, 1992 the Department of Defense issued a document entitled "DoD Software Reuse Initiative Vision and Strategy," based on the premise that system architectures, designs, test plans and software can be "reused" as part of new systems development in order to improve the quality and reliability while lowering the cost of software intensive systems. In establishing the framework for this reuse effort, DoD set four specific goals (DoD Reuse Executive Steering Committee, 1992):

- Increase software quality and reliability.
- Improve the management of software technical risk.
- Shorten system development time.
- Increase productivity.

Reuse refers to reusing existing software. To achieve its goals, DoD has taken the approach of integrating reuse into the software development process. As part of its strategy to measure progress towards these goals, DoD has established a

pilot metrics program which outlines data to be collected in all software reuse activities in order to support its management and control objectives. Specifically, DoD proposes that metrics be defined, collected, and analyzed in order to measure the degree of reuse success (DoD Reuse Executive Steering Committee, 1992).

Presently, a software reuse metrics plan has been developed for DoD; Chapter IV references the contents of this plan as part of its discussion on establishing a software quality measurement program.

C. CURRENT DOD EFFORTS

To establish the foundation for reuse employment, a number of federal and civilian agencies have embarked on an effort to build "libraries" of reusable software components that can be retrieved and integrated into new systems development. The success of their effort depends, in part, on two things; first, the library must collect reusable components that will satisfy the requirements of major library users; and secondly, these components must be of the highest quality. As Sommerville (Sommerville, 1992) suggests, a successful software reuse library must satisfy four customer requirements:

- It must contain software of value to the customer.
- The software and documentation must be understandable.
- The customer must be confident using the software.

- The software must include information on its reuse.

Of interest in this thesis is the third requirement. The best collection of reusable software is useless unless the customer is confident that by using them he will realize some quantitative benefit (e.g., increased productivity, reduced development costs or improved system quality).

Today, all services as well as other select government and non-governmental agencies have operational reuse libraries and are developing and implementing reuse practices. Among the services, all report a reduction in systems development and testing time with associated increases in system quality where reuse practices have been implemented (Foreman, 1993). One of the key governmental agencies involved in this effort is the Defense Information Systems Agency (DISA).

In 1992, DISA, with the support of the Joint Interoperability Engineering Organization (JIEO) and the Center for Information Management (CIM), established its Software Reuse Program to serve as a prototype for the DoD-wide reuse initiative (DISA/JIEO/CIM Software Reuse Program, 1993). This thesis will look at DISA's effort to support DoD's reuse vision. Specifically, it will discuss DISA's software reuse library management and will introduce a methodology for the collection and analysis of metrics relating to software performance in order to improve library software quality.

While reusable software can include any of a variety of programming languages, this study is directed towards Ada software components for the following reasons:

- Ada is the standard DoD systems development language; hence, it is the logical language on which to base any software quality improvement efforts.
- DISA has developed specific measurement and certification processes for Ada software and has automated tools for Ada source code analysis.

Chapter II continues this discussion.

II. DISA's REUSE PROGRAM

A. INTRODUCTION

The reasons supporting the development of a library of software reusable components were outlined in Chapter I. As discussed, DISA along with JIEO and CIM established the Software Reuse Program as a means to support DoD's reuse initiative. Presently, DISA operates its own reuse library, the Defense Software Repository System (DSRS), which serves as a repository of software assets for use by DoD customers. Additionally, DISA supports a number of distributed Software Reuse Support Centers (SRSCs) which have agreed to provide local support for the integration of reuse practices throughout DoD (DISA/JIEO/CIM Software Reuse Program, 1993).

DISA's main objective for its DSRS is to provide a source of reusable software components, called ***Reusable Software Components*** (***RSC***s) in DISA terminology, for use by program and domain[1] managers in their systems development efforts. RSCs include products from all phases of the software development cycle to include such items as requirements, design and testing documentation, as well as source code and manuals.

---

[1]A set of software systems with common features and functionality. The set may be horizontal (e.g. aircraft navigation system) or vertical (e.g. radar software) (Ogush, 1992)

The process of building a library of reusable components begins with a domain analysis study involving major potential library users in order to determine the type and identity of assets they need. Based on domain analysis results, the library starts collecting candidate RSCs, evaluating their reusability potential, certifying them and installing them in the library for use (DISA/CIM Software Reuse Program, 1993). This process is outlined briefly in the following sections. While this discussion is general in nature, specific references to Ada RSC processing are made occasionally in order to foster a better understanding of the theme of this thesis.

B. RSC PROCESSING OVERVIEW

Reuse library development begins with a study of customer needs. Through domain analysis and liaisons with major potential library users, the library determines which assets will satisfy user requirements. Once specific asset types are identified, library efforts focus on collecting and cataloging these RSCs. As previously stated, RSCs include not only source code but also supporting design, development and testing documentation.

RSC collection is an ongoing process that relies on both government and private industry sources for component contributions. Cataloging these components for library use involves identifying RSCs with potential for reuse, analyzing

their code and documentation, and then certifying them accordingly. The net result of the cataloging process is a certified RSC ready for induction into the reuse library. Because the process of collecting and analyzing candidate RSCs is lengthy in nature, it is not discussed in detail here. Rather, it is the RSC certification process that is the focus of this study.

C. RSC CERTIFICATION

Between the collection and certification phase, each RSC is analyzed and evaluated to determine if it meets the criteria for certification or if it requires re-engineering to bring it up to library specifications. If re-engineered, the RSC is subjected once again to the analysis and evaluation process. Once this process is complete and documented, the component is ready for certification where certification involves assigning a level of completeness to the RSC based on the following schema (DISA/CIM Software Reuse Program, 1993):

- Level 1 - Completeness and functionality of RSC are unknown. No measures of quality are provided.
- Level 2 - RSC completeness is assured. Code compiles if provided. No testing or user manual given.
- Level 3 - RSC is complete and complies with reusability criteria. Testing occurs and results are provided.
- Level 4 - RSC is complete, meets reusability requirements, is tested and user manual is provided. Highest degree of confidence in RSC quality.

RSC processing concludes with the installation of the RSC and its supporting documentation into the reuse library. While DISA's RSC certification process provides useful information to the library user, it has its limitations.

D. CERTIFICATION LIMITATIONS

As described above, RSC certification levels indicate the amount of supporting documentation available for the RSC rather than a quantifiable measure of expected quality (Merritt, 1993). For Ada RSCs, some indicative measures of quality are derived from the statistical metric data collected on the actual software code using the automated analysis tool AdaMAT/D. While this type of analysis may provide some general indication of quality, a more quantitative approach is desirable in order to satisfy customer expectations.

E. CUSTOMER EXPECTATIONS

In theory, a customer of a software reuse library expects, either implicitly or explicitly, a certain degree of quality from a RSC he draws from the library. This idea is embodied in the characteristics identifying a successful reuse library as outlined by Sommerville (Sommerville, 1992). While static measures, such as those collected by AdaMAT/D, provide a general qualitative basis for determining expected quality, they are not sufficient to predict software behavior once that software is subjected to a variety of domains and operating

environments. Customer expectations coupled with DoD initiatives for the reuse of software in DoD projects make the issue of quality a realistic concern.

As a first step towards meeting customer quality demands, this thesis proposes the development and integration of a metrics program which will support quantitative quality management of reusable software components. Chapter III discusses this proposal in detail.

III. TOWARDS BETTER QUALITY

A. BACKGROUND

Software quality can be discussed primarily in two contexts: the *process* and the *product*. Dunn and Ullman (Dunn and Ullman, 1982) make the observation that in the 1970s, the software industry often perceived quality in the context of the product, as a post production inspection function; this perception still exists in many DoD organizations today. For many DoD contractors, the opposite is true; perhaps no single factor of software development is given more of their attention than quality improvement in the production process. Driven by current economic conditions, DoD no longer has the luxury of developing and implementing software systems with a high probability of defects and associated high maintenance costs to fix them. Rather, DoD must now ensure that the highest standards of quality are applied throughout the production process in order to minimize the risk of poor quality in the finished product.

B. THE QUALITY OBJECTIVE

It is reasonable to assume that the successful achievement of quality improvement in software products lies in the ability of DoD and industry to take a proactive role in managing the development process. In their text on software

reliability, Musa et al. (Musa et al., 1987) identify three user-oriented characteristics related to the software product: ***cost***, ***schedule*** and ***quality***. As they point out, of these characteristics, quality is the only aspect of a product that cannot be given a quantitative measure. They go on to suggest that ***reliability*** is an intrinsic characteristic of quality that subsumes many of the other properties normally associated with the term quality.

To the user, software reliability means that a given program will operate for a period of time without failure; or conversely, that a program behaves as intended for an indefinite period of time. Musa et al. believe that, because reliability relates to the operation of software, it most appropriately supports the user's idea and view of software quality. For that reason, they propose that reliability measuring:

- Is customer, rather than developer oriented.
- Relates to the operation rather than design of software.
- Accounts for the frequency of problems.
- Is suitable for predicting trends.

This discussion concludes by suggesting that reliability measurements (e.g., time to failure, failure count) can play an integral role in the movement towards the quality objectives of the reuse initiative. This role and its application in the reuse library environment are discussed in

the following section. Note that for the remainder of this thesis, the term ***reliability*** is used as a specific connotation of the more general term quality.

C. THE REUSE LIBRARY'S ROLE

Because the reuse library is primarily a repository of software components, its part in facilitating quality improvement may not be understood. Clarity on this issue may be gained by discussing the present and potential roles of the reuse library in meeting quality objectives.

1. PRESENT ROLE

As discussed in Chapter II, DISA currently manages RSC quality through its component certification process which provides the user with only an indication of the component's level of completeness. For Ada components, a qualitative measure of quality is provided through data collected as part of the static analysis done on the code; in any case, the user gets no guarantee of the software's behavior once it is placed in operation.

2. POTENTIAL ROLE

DISA can expand its present role in providing quality software by adding a metrics program to monitor and evaluate the reliability of software before and after it is fielded in an operational environment. Presently, a number of organizations in the software industry have successfully developed and implemented programs which support this type of

quality analysis. This thesis will draw on the best of those methodologies to develop a similar program for the reuse library at DISA.

D. SOFTWARE QUALITY-PROGRAM PROPOSAL

The purpose of this thesis, as stated earlier, is to outline a methodology, based on the successful efforts of other organizations such as NASA, for collecting, analyzing, and applying operational and metric data from reusable software components in order to establish a quantitative basis for predicting their reliability. Again, this effort is directed towards Ada software with the added limitation of applicability to stand-alone, functional RSCs as opposed to integrated application programs.

E. THESIS OBJECTIVE

The objective of this study is to propose a plan by which DISA can broaden the role of software quality management in its reuse libraries; this plan includes procedures for collecting and analyzing user reliability data on reuse components. Ideally, this plan will provide a supplementary link to the current library management process that will enable library asset managers to better quantify and predict the reliability of library software over time.

As a way of introduction, the idea behind this plan is to develop a quality measurement methodology that will support

the capture and analysis of software metrics as well as operational data in order to produce some quantitative measure of expected quality, or more specifically, reliability. Figure 3.1[2] provides an overview of this methodology.

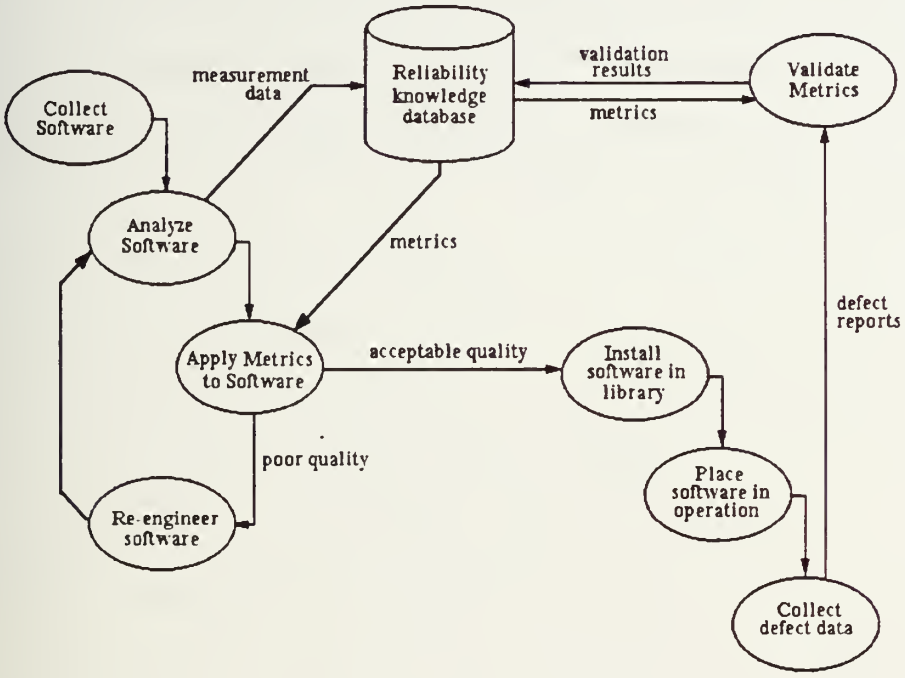

Figure 3.1 Overview of a Software Quality Measurement Methodology

As Figure 3.1 illustrates, at the core of this methodology is a reliability knowledge database which serves as a receptacle for RSC measurement and operational (defect) data. As the reuse library and reliability knowledge database mature, software quality managers will be able to apply past

[2]Adapted from an illustration by Sheldon et al. (Sheldon et al., 1992)

experiences with software reliability to the analytical processing of new library components.

As this figure shows, this methodology is not intended to be a static, one-time application analysis; this type of analysis currently exists. Rather, it will serve as a basis for a continual, evolutionary program for improving software quality. The development of a quality measurement program to support this study's objective is the subject of the remainder of this thesis.

IV. DEVELOPING A SOFTWARE QUALITY MEASUREMENT PROGRAM

A. BACKGROUND

As discussed in the previous chapters, improving the reliability of software is an integral part of improving overall software quality; and, the key to improving reliability is the establishment of a methodology to capture and analyze metrics which support quality management. Schneidewind[3] (Schneidewind, 1993), in his discussion on the methodology of metrics, outlines five steps to follow when implementing a software quality measurement program:

- Define software-quality requirements.
- Select potential software-quality metrics.
- Design and implement a metrics plan.
- Analyze metrics data.
- Validate original software-quality metrics.

Schneidewind points out that these steps form the basis for an iterative process which involves analyzing and adjusting measures as needed.

---

[3]Dr. N. F. Schneidewind has worked extensively in the field of software reliability modelling. He is the developer of the Schneidewind Software Reliability Model used by IBM-Houston to predict software reliability for the NASA Space Shuttle flight software.

David Siefert (Siefert, 1989), who has researched the implementation of software reliability measurement programs in organizations, contributes a model of this process. Figure 4.1 (adapted from Siefert's model) combines the ideas of Schneidewind and Siefert to provide a graphical process representation.

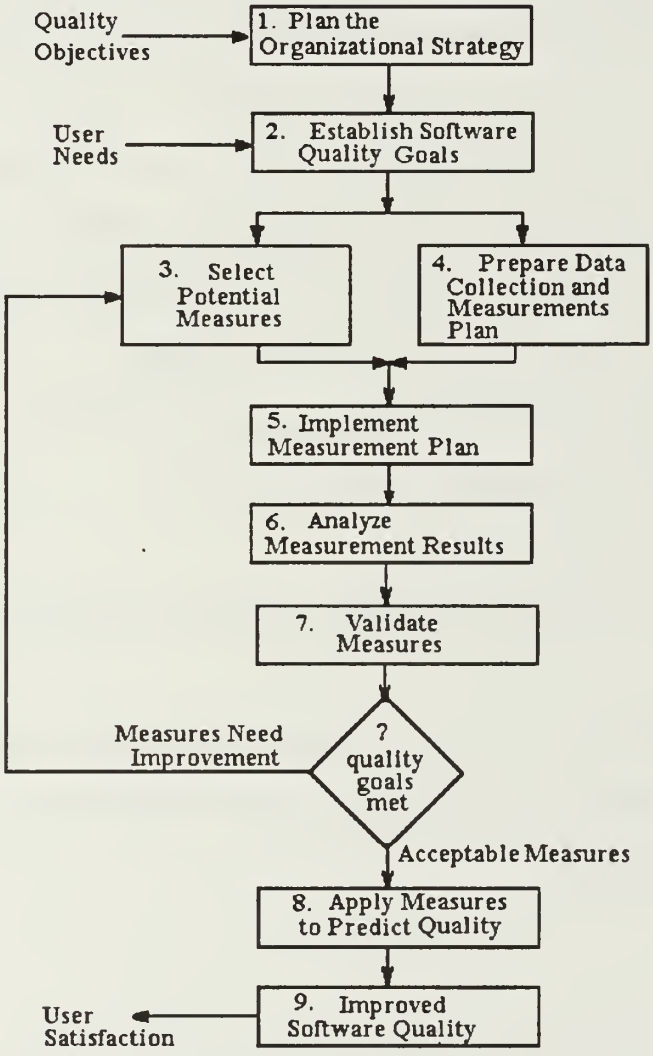

Figure 4.1 Quality Measurement Program Methodology

This measurement methodology, which adheres to IEEE standards, provides the framework for a discussion of DISA's organizational goals and the development of a software quality measurement program to support these goals. However, before doing so it is necessary to look at DISA's current measurement efforts.

B. DISA'S CURRENT SOFTWARE REUSE METRICS PROGRAM

To manage its software reuse program, DISA, in concert with JIEO and CIM, developed their *Software Reuse Metrics Plan* (DISA/JIEO/CIM Software Reuse Program, 1993). This plan, designed in response to DoD's proposal to define metrics which can be used to measure reuse success (DISA/CIM Software Reuse Program, 1993), outlines the requirements for identifying, collecting and reporting metrics for management analysis of the DoD software reuse program. Specifically, it provides Project Managers, Domain Managers, Repository Managers and DoD Executives with a process that will enable them to measure and manage software reuse in their area of responsibility.

Due to its broad scope of application, this plan focuses on high level process metrics rather than on the product metrics suitable for the software quality analysis focus of this study. Therefore, this thesis will use IEEE standards and the methodology diagramed in Figure 4.1 to outline a new, supplementary program for quality measurement in the reuse

library environment. The framework for this program is discussed next.

C. PROGRAM FRAMEWORK

Figure 4.1 diagrams the methodology for establishing a software quality program. The remainder of this chapter focuses on the first three steps in this methodology by discussing an organizational strategy for DISA as well as software quality requirements and supporting metrics. Chapter V continues this discussion by addressing the metric collection process. Chapter VI completes this study by addressing the analysis and use of operational data as it relates to improving software quality in the reuse library environment.

1. ORGANIZATIONAL STRATEGY

DoD, as part of its organizational strategy to implement systematic reuse (DoD Reuse Executive Steering Committee, 1992), called for the establishment of metrics collection procedures to measure reuse effectiveness. DISA, in support of this DoD directive, developed its software metrics plan (DISA/CIM Software Reuse Program, 1993). Contained in this plan are the elements of its three-phase organizational strategy; these phases, listed in decreasing priority, are:

- Phase I - Focus on developing the reuse library.
- Phase II - Examine the cost and benefits of library use.

- Phase III - Address the technical aspects of reuse.

Phase III of DISA's strategy targets the technical issues of reuse. For the reuse library manager, this means identifying which design quality metrics are most useful in providing the best indication of software's suitability for reuse (DISA/CIM Software Reuse Program, 1993). Therefore, one element of DISA's organizational strategy can be stated as follows:

- *To improve the quality of the software stored in the reuse library.*

Based on this strategy, organizational software quality goals can be developed.

2. GOAL IDENTIFICATION

Before software metric selection can be considered, an organization must first identify its quality requirements. For DISA, a general quality requirement is that its library offer RSCs that customers can integrate into their applications to reduce costs and development time. According to IEEE (IEEE, 1992), quality requirements should be expressed in one of two forms:

- ***Direct metric value*** - a quantitative value which provides a direct measure of some characteristic of software quality[4].

[4]For instance, ***defect-report-count*** might be used as a direct measure of RSC reliability.

- ***Predictive metric value*** - a quantitative value used to predict some characteristic software quality[5].

For this study, ***reliability*** has been identified as the characteristic of software quality of interest. Ideally, the use of direct metrics is desirable; however, this assumes that this type of data is available. In the reuse library environment, this idealization is met when a RSC is received and has been thoroughly tested to gather this information. In other cases, when testing is incomplete or the validity of the testing is in question, predictive metrics can be used.

For the reuse library, direct metrics appear to be the most suitable form for expressing quality requirements based on the fact that the library's primary role is that of a software repository and not a software development facility. However, this assumption in no way precludes the use of predictive metrics. In fact, as will be discussed later, predictive metrics that have been validated can serve as valid indicators of software reliability until direct metrics are available; application of both metric types is addressed in this study.

As illustrated in Figure 4.1, the development of a good metrics program is an iterative process where metrics are selected, applied and evaluated for suitability. In the

---

[5]For instance, ***number-of-statements*** might be used as a predictor of RSC reliability.

interest of establishing a baseline for the start of this process, the proposed goal for DISA's reuse library can be formally stated as follows:

- *To achieve a zero defect-report-count for library reusable software components.*

A prerequisite for achieving this goal is that a particular software component has undergone rigorous inspection and testing; in reality, a zero *defect-report-count* is possible for a component that was never tested. The next section will discuss metrics to support this goal.

3. METRICS SELECTION

Before embarking on metrics selection, it is important to first define what a metric is and what it does. Reindollar (Reindollar, 1993) suggests that a metric is a tool to be used by managers to determine their progress towards meeting a specified goal. In the abstract sense this is true but a more formal definition is desirable. Recall that a metric can be either direct or predictive. While the definition of a direct metric is straightforward, a more detailed definition of a predictor metric is desirable.

Schneidewind defines a predictive metric[6] as a function that inputs software data and returns a single numerical result. He uses cyclomatic complexity as an example

[6]Schneidewind uses the term "quality metric" in his writings.

where the formula $M = e - n + 2$ is the metric function. In this function, e (edges) and n (nodes) represent software input data and the resultant value M represents the output. The significance of this definition is that the single numerical output of the metric function allows the user to compare a software component or module against a standard. For example, if $M_1 = 2$ for module one and $M_2 = 3$ for module two, there exists a quantitative basis for comparative analysis of the two modules if M has been previously validated against a quality factor like reliability. Finally, provided a predictor metric is valid (as will be discussed), it can serve as a substitutive approximation of the desired quality characteristic. This feature of a predictive metric is particularly desirable during the software development process where the characteristics of quality, such as reliability, can not be determined until development completion. (Schneidewind, 1992) .

With the concept and application of metrics better defined, the determination of the metrics suitable for DISA's quality measurement program can be made. While any number of metrics, such as complexity metrics, can be used as indicators of reliability, the following two metrics have been selected for purposes of illustration in this study:

- Direct Metric - ***defect-report-count*** (a count of all discrepancies related to any portion of a RSC)
- Indirect Metric - ***number-of-statements*** (the total statement count of the code portion of a RSC)

One reason for illustrating the methodology with ***number-of-statements*** is that, in the final analysis, many other metrics have been shown to be highly associated with program size.

As mentioned, more than one metric of each type are suitable for use. In fact, Siefert (Siefert, 1989) in his research identifies and ranks 15 "Best of Class" metrics based on frequency-of-use, importance, ease-of-use and ease-of-implementation as reported by the software industry. He suggests that an organization developing a quality measurement program select two or three of these metrics based on their meaningfulness to the organization. A list of these metrics is provided in Appendix A.

Figure 4.2 (adapted from IEEE) depicts the hierarchical relationship between software quality, quality factors and direct and indirect metrics.

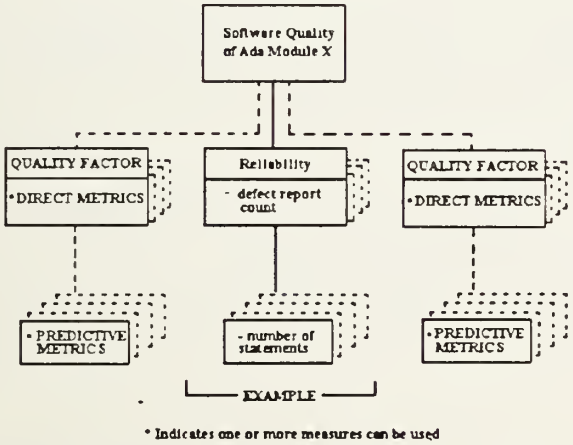

Figure 4.3 Metrics Hierarchical Tree (IEEE, 1992)

Examination of this hierarchical tree shows that software quality is composed of a number of quality factors where each factor has one or more direct metrics representing it. Connected with each quality factor is one or more predictive metrics which serve as substitutes for direct metrics when they can not be used. The reader is directed to IEEE's standard on quality metrics (IEEE, 1992) for further information on the subject.

D. METRICS IMPLEMENTATION, ANALYSIS AND VALIDATION

With suitable metrics selected to support software quality management, the next step in the measurement program methodology is the implementation and subsequent analysis and validation of these metrics. Chapter V continues with program implementation.

V. PROGRAM IMPLEMENTATION

A. BACKGROUND

At this point, the need for improving software quality has been identified. Likewise, DISA's role in developing a reuse library to support DoD software quality initiatives has been discussed. Chapter IV suggested a software quality goal for DISA as well as a framework for establishing a reliability measurement program to support that goal; this chapter addresses the program's implementation. Before continuing, it is useful to establish a clear understanding of the terminology that will be used.

B. TERMINOLOGY

For clarity in this discussion, the following IEEE software definitions are provided (IEEE, 1990):

- ***Error*** - a logical, syntactic or clerical discrepancy introduced in the software during the design process.
- ***Fault*** - an unintended functioning of software due to one or more errors.
- ***Failure*** - unexpected results from software as a consequence of one or more software faults.
- ***Defect*** - for purposes of this thesis, ***defect*** is synonymous with ***fault***.

In the software reuse environment, the greatest potential for defects exists either during the interfacing or adaptation

of reusable software. RSCs are similar to commercial off-the-shelf software products; therefore, their successful use is dependent upon the user's understanding of their interface requirements and intended applications. Since the interest of this study lies in minimizing the risk inherent in the RSC itself, defects related to user misunderstanding of the RSC are generally not of interest. The next section continues with a discussion on data collection.

C. THE NEED FOR COLLECTING SOFTWARE DATA

To support a software metrics program, two types of data are needed: metric data relating to the software itself (e.g., ***number-of-statements***) and defect data relating to the operation of the software (e.g., ***defect-report-count***).

For this study, there are two[7] specific reasons for collecting software defect data; to support direct metric assessment of software quality and to validate the suitability of predictor metrics. For example, ***defect-report-count*** can be used directly by the ***Software Quality Manager*** (***SQM***) to discriminate between good and poor reliability software. On the other hand, ***defect-report-count*** might be used to validate the predictor metric ***number-of-statements*** so that ***number-of-statements*** can be used to predict reliability when the actual

---

[7]A third reason is for high-level, managerial trend analysis. The reader is directed to Florac's report (Florac, 1992) for further details.

number of defects in a component is unknown. Therefore both metric and operational data are needed to provide the SQM with the ability to develop more than one means of evaluating software quality.

In concluding this discussion, it is worthwhile to point out that Musa et al. (Musa et al., 1987) recommend collecting all data, particularly failure data, during software's operational phase. This requirement for good data collection is outlined in the following sections.

D. PREREQUISITES FOR DATA COLLECTION

The previous section outlined the need for data collection to support a metrics program. Therefore, the first consideration for data collection is to identify the exact data requirements to support each metric being used. Other considerations are: data collection responsibility, data collection tools and data storage (IEEE,1992). Each of these considerations warrant further discussion.

1. DATA REQUIREMENTS

For defect data, Keller[8] suggests that an extensive database of software defect data is essential for reliability analysis (Keller, 1993). To that extent, he recommends collecting at least the following software defect data:

- Time when failure occurred or was detected.

---

[8]Keller lends extensive experience in performing software reliability analysis on Space Shuttle software to this study.

- How the failure was found: static analysis, testing or operational use.
- Nature of the failure: code fault or human error.
- History of the fault introduction that caused the failure.
- Conditions or environment which triggered the failure.
- Why the failure was not detected earlier.
- The effect (severity) of the failure.
- Configurations or versions of the software affected.
- Action taken to correct the failure.

While this list is not inclusive, it does provide a framework for initial data collection. In addition to Keller's recommendations, Basili and Weiss (Basili and Weiss, 1984) suggest including the user in the data requirements discussions. By doing so, they believe that end-user viewpoints and complaints can be acknowledged early in order to make them feel a part of the data collection process. Appendix B concludes this discussion by providing a suggested sample defect report for DISA's use.

For metric data, requirements are straightforward; for each software metric selected, collect all of the data relevant to its use. For example, for the metric ***cyclomatic complexity*** discussed in Chapter IV, the SQM will want to collect information on the number of edges (e) and nodes (n). Data collection for other metrics is done likewise.

2. DATA COLLECTION RESPONSIBILITY

The Reuse Library, which analyzes and re-engineers software components, is the logical metric data collecting organization. While the argument might be made that the software developer is in a better position to collect this information, care must be exercised by the library to ensure accurate data is collected to prevent GIGO[9]. For defect data, the library will need to rely on the user for timely and accurate collection. Industry practice is to use "*Defect Reports*" for this type of feedback. The reuse library is then responsible for providing the user with feedback reporting forms to gather this information. As stated earlier, the success of the metrics effort relies, in part, on the cooperation of users; this matter can be dealt with as a library policy issue.

3. DATA COLLECTION TOOLS

As with data collection responsibilities, the choice of data collection tools depends on the category of data. For software metric data, a number of automated tools exist which can collect needed information. Further discussion and recommendations on such tools is beyond the scope of this study. For defect data, perhaps the best collection tool is the defect reports mentioned earlier. For metric data, analyzers can capture code metrics during compilation.

---

[9]Garbage In, Garbage Out.

4. DATA STORAGE

For data storage, some type of electronic storage facility is desirable. For example, a database would provide a means of storing both metric and defect data on software components. Electronic storage can also support data retrieval and analysis. As with data collection tools, it is the library's responsibility to establish a data storage mechanism suitable to its needs.

E. OTHER CONSIDERATIONS

Developing a plan is the first step in the data collection process. The previous section discussed the elements of such a plan. However, the plan is not complete without addressing the following considerations.

1. MISINFORMATION PITFALLS

One important requirement in data collection is data exclusivity. Florac warns that care must be taken to ensure that the data items collected are mutually exclusive of one another to avoid duplication in reporting (Florac, 1992). A second requirement is the use of a good, communicative tool. As mentioned, a data collection tool (e.g. a defect report) must provide an effective means of communication between the user and the reuse library. To do this, it must be designed in such a way that it is unambiguous, not subject to interpretation and not redundant.

2. NECESSITY OF DATA VALIDATION

Basili and Weiss (Basili and Weiss, 1984) found in their research on developing a data collection methodology that data validation is a necessity for a good data collection program. They point out that patterns of mistakes and misclassifications in data reporting become evident as the collecting agency begins to synthesize and validate user feedback. To counter these inaccuracies they suggest the use of interviews with defect reporting activities to clarify any potential misunderstandings. As a final note, they warn against data entry errors when automated databases are used; such errors will unknowingly skew the data.

3. REPORTING PITFALLS

As is well known in the software industry, quality testing only proves that software meets certain criteria; it in no way guarantees the absence of defects. The same is true of data reporting. As Musa et al. point out (Musa et al., 1987), lack of defect data must be given the same concern as its actual presence for the reason that defects often go either undetected or unreported.

Although Appendix B provides a sample defect report, this report serves as only an example and may be subject to modification based on its usefulness and appropriateness in collecting the necessary data to support the reuse library's

metrics program. Chapter VI continues by discussing the analysis and validation of data, once collected.

VI. DATA ANALYSIS AND METRICS VALIDATION

A. INTRODUCTION

As Schneidewind points out (Schneidewind, 1993), metrics provide a quantitative rather than qualitative basis for evaluating software quality. In its discussion on metrics, Chapter IV identified two types of metrics that are of interest in this study: direct and predictive.

To be used effectively, each metric needs an associated acceptance criterion (threshold value) which distinguishes good from poor quality. For example, predictive metrics are collected on software during its development; by comparing these metric values to the threshold value, the developer can determine whether or not quality development goals are being met. Likewise, direct metrics are collected during software testing and operation; again, these values are compared to an acceptable threshold value to determine final product quality. The key then to a successful metrics program lies both in the choice of metrics and metric thresholds; metrics and metric thresholds are only as useful as their ability to indicate whether or not software quality requirements are being met (Schneidewind, 1993).

While direct metrics serve as unambiguous discriminators of software quality, the value of predictor metrics is

initially unknown; therefore, validation of predictor metrics is necessary before they are applied. This chapter will examine the use of metrics in determining the degree to which software quality goals are being met and the process of predictive metrics validation.

B. METRIC DATA ANALYSIS

A metric data collection program can serve as a valuable tool to the software quality manager in the reuse library. Predictive metrics can aid the manager in determining which RSCs are most likely to be suitable for library use. Direct metrics can be used to identify substandard RSCs and to support the evaluation of the library's metrics plan. The use and interpretation of each of these metrics is presented below.

1. DIRECT METRICS ANALYSIS

As defined in Chapter IV, the direct metric of interest in this study is the *defect-report-count* for a particular software component. As noted, before metric analysis can begin, some threshold evaluation criteria must be established. While an ideal goal is to achieve a zero *defect-report-count*, a less stringent criteria may be more practical; choice of this critical value is left up to the organization based on its quality requirements and past experience. Further guidance is provided in Appendix A which

lists 15 common industry metrics and suitable threshold values based on industry experience.

Once an evaluation threshold is established, a software component can be measured against it. For example, given candidate RSCs with a *defect-report-count* of two and a pass-fail threshold value of two, a reuse library certification team has three options: accept the RSC for certification; mark it for re-engineering; or reject it. Or, consider the RSC that has been certified and installed in the library. As user *defect-report-count* data is collected, the certification team can at some point re-evaluate the component based on this data and the established criteria. As with any metric, the appropriateness and validity of the direct metric being used is important and should be subject to evaluation.

2. PREDICTOR METRIC ANALYSIS

As identified in Chapter IV, *number-of-statements* is a sample predictive metric used in this study. While direct metrics are used for software product analysis, predictor metrics are used during the actual software development process itself. As with direct metrics, some threshold evaluation criteria must be established for predictive metrics before analysis can begin. Once this is done, metrics are collected on software at specific intervals during its development; these metrics are then compared to the evaluation criteria to determine if the software is being developed

within the guidelines determined result in good final product quality.

Predictive metrics can be applied in the reuse library in a number of ways. For example, take the RSC that has been marked for re-engineering. In this case, predictive metrics can be applied during the re-engineering process to ensure it is designed to meet better quality standards. Another example is that of RSCs which are certified at Level One or Two or for which no test data is available. Here, predictive metrics can be collected and used to provide some quantitative indication of the RSC's reliability potential.

While direct metrics provide an unambiguous measure of reliability (either the software has failed or it hasn't), predictor metrics can do only that: predict. For this reason, the critical step of predictor metric validation is necessary in order to establish their appropriateness as reliability indicators (Schneidewind, 1992). The next section discusses metrics validation.

C. PREDICTOR METRICS VALIDATION

As Schneidewind points out (Schneidewind, 1992), the purpose of metrics validation is to establish a high degree of association between a metric and the quality factor it represents. Since the predictor metric ***number-of-statements*** is used to represent the direct metric ***defect-report-count*** when it is unknown, it is necessary to validate ***number-of-***

statements to ensure that there exists a strong association between it and *defect-report-count*. The validation of metrics presupposes that two prerequisites are in place before the process begins: first, a sound methodology for metrics validation; secondly, sufficient data to make the validation results reliable.

The first step in the validation process then is to establish a criteria against which metrics can be validated. Schneidewind (Schneidewind, 1993) provides the following criteria based on IEEE standards[10]:

- ***Correlation*** - The variation in *defect-report-count* must be strongly associated with the variation in *number-of-statements* for a given software component.
- ***Tracking*** - A change in *defect-report-count* must be accompanied by a directly proportional and positive change in *number-of-statements*.
- ***Consistency*** - If *defect-report-count* is rank-ordered for a given set of software components, *number-of-statements* for those components must have the same ordering.
- ***Predictability*** - If *number-of-statements* is to be used as a predictor of reliability, it must be able to do so within a given accuracy.
- ***Discriminative power*** - *Number-of-statements* must be able distinguish between high and low reliability software.
- ***Reliability*** - *Number-of-statements* must meet all of the following criteria a given percentage of the time: correlation, tracking, consistency, predictability and discriminative power.

[10]The quality factor and metric examples from this study are used in these definitions for clarity.

A given metric does not have to satisfy all criteria. Rather, the metric must satisfy those criteria that are related to the applicable quality functions (see the forthcoming "Sample Analysis" discussion).

With a validation criteria established, the validation process, as outlined by IEEE's (IEEE, 1992), can begin. This process, consisting of drawing a sample of RSC data, conducting a statistical analysis of the data and recording the results, is discussed in the following sections.

1. DATA SAMPLE COLLECTION

As discussed in Chapter V, some means of storing software metric and defect data are assumed to be available. For example, Chapter III illustrated the use of a reliability knowledge database (Figure 3.1) which could serve as an information repository from which a representative sample of RSC metric and failure data would be drawn.

2. SAMPLE ANALYSIS

Once a representative sample of RSC data is collected, the next step is to test the data with respect to the validity criteria outlined above. First of all, Schneidewind (Schneidewind, 1992) defines three quality functions to which the validity criteria apply:

- ***Quality Assessment*** - criteria used by software quality managers to perform a relative (ranking) comparison of software quality in a set of components.

- ***Quality Control*** - criteria used by software quality managers to distinguish components with acceptable and unacceptable quality via discriminative value analysis.
- ***Quality Prediction*** - criteria used by software quality managers to forecast the quality of components and to flag those not meeting requisite standards.

According to IEEE standards (IEEE, 1992), a metric may be used for quality analysis only in the areas for which it passes the validity test. Schneidewind adds that the validity criteria against which a metric is tested depends on the quality function requirements. For example, a RSC certification team may be interested only in validating discriminative metrics while the re-engineering team may consider validating only predictive metrics. In any case, the organization must determine the breadth of metric-to-criteria validation. Schneidewind (Schneidewind, 1992) and IEEE (IEEE, 1992) both provide descriptive examples of tests which can be conducted on a metric to prove its validity with respect to each of these criteria; the reader is directed to Appendix C for further information.

Metric validation concludes when all requisite tests are complete and the appropriate statistical data is collected. As illustrated at the conclusion of Step 7 in Figure 4.1, at this point the software quality manager can evaluate the appropriateness of the original metrics selected for organizational use. Metrics having failed or performed poorly in some tests can be eliminated and replaced with other

more suitable candidates and the process of data collection and validation repeated.

3. RESULT DOCUMENTATION

At the conclusion of metrics validation, all results of the tests on the predictive metrics are recorded. As noted before, the entire software quality measurement program is an iterative process where predictive metrics are chosen and applied to a software project. From there, operational and test data is collected in the form of direct metric values. These direct metric values are then in turn used to validate the suitability of the original predictive metrics.

While the distinction between good and bad predictor metrics can be made on the basis of the validation results, the one-time validation of a metric does not guarantee its future effectiveness; certain considerations are relevant to any metric validation (IEEE, 1992):

- **Need for re-evaluation** - metrics validated in one environment or application may not be valid in another or if invalidated in one environment or application, may be valid in another.
- **Confidence in the validation** - as the use of the metric increases and the same predictable results are achieved, confidence in the metric will grow.
- **Environment** - validated metrics should be applied in the same environment as which they were validated to ensure best predictive ability.

At this point a reliable set of metrics is assumed to exist for an organization. Because this discussion of the

validation process was general in coverage, Chapter VII provides a case study to illustrate the entire metrics program methodology. The final action on the part of the SQM is to apply the newly validated metric to a project. A discussion of this metrics application is included before closing this chapter.

D. METRICS APPLICATION AND CONCLUSIONS

As discussed in the metrics validation section above, both direct and predictive metrics play an important role in managing software quality. Of the two metrics, direct metrics are the most useful in discriminating between good and poor quality components. In the reuse library environment, direct metrics can be collected as the RSC is tested or after it is placed in operation by the user.

Predictive metrics, although less desirable, are useful when direct metrics are not known. Through the process of validation, predictive metrics can be shown to be reasonably accurate in discriminating between good and poor quality components. As with direct metrics, predictive metrics can be collected in the reuse library during the process of analyzing a RSC for certification. At that point, the certification team can use the resultant measures to predict if the RSC will meet operational software quality standards. If not, the component can be re-engineered where predictive metrics will be used to guide the process.

As mentioned and illustrated throughout this thesis, metrics can play a key role in any organization's software quality program. Metrics alone are not a solution to the reuse quality problem. Rather, they are a tool to be used prudently by the software quality manager to manage and improve the quality of organizational software. In concluding this study, Chapter VII provides an actual example of a metrics program in practice.

CHAPTER VII. A QUALITY MEASUREMENT PROGRAM CASE STUDY

A. INTRODUCTION

Chapter III outlined the need in the reuse library environment for a quality management program. Figure 3.1 provided an overview of how such a program could fit into DISA's current operations. Chapter IV then provided a methodology for this program and discussed the framework for its implementation. Chapter V continued by developing a program implementation plan which discussed the requirements for data and the means for collecting it. Chapter VI concluded with a discussion on the use of software metric data and the need for its validation. This chapter presents a case study to tie together the discussions of Chapters III through VI.

This case study is based on research conducted by Norman F. Schneidewind whose work has been referenced extensively in this thesis. Schneidewind was chosen for the reason that his theories for software quality measurement have proven to be reliable in real-life practices. A primary example is the successful application of his measurement methodologies to Space Shuttle avionics software in order to measure and predict its quality (Schneidewind and Keller, 1992). The remainder of this chapter provides an overview of

Schneidewind's rationale for using software measurements to predict quality as well as the means to validate the usage of those measurements.

B. SOFTWARE QUALITY MEASUREMENT PROCESS OVERVIEW

While Chapter III outlined a quality measurement methodology and Chapter VI discussed the uses of metrics in analyzing and predicting quality, this chapter provides a detailed discussion of the "why" and "how" associated with software quality measurement.

Schneidewind, in his discussion on metrics validation, provides a model of the process and a description of the terms used in metrics collection and validation; Figure 7.1 illustrates this model.

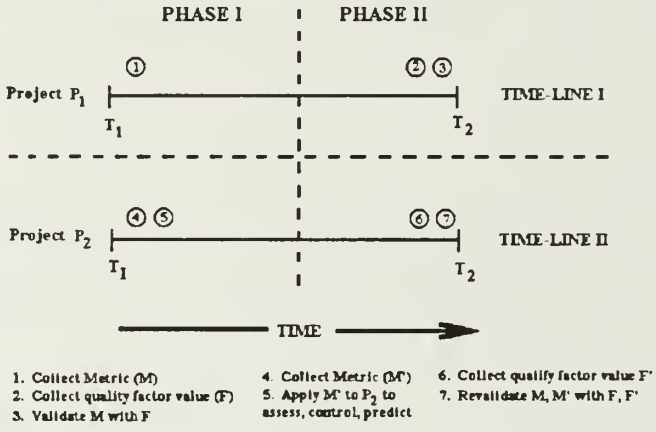

Figure 7.1 Metrics Validation Process (Schneidewind, 1992)

As shown in this figure, a metrics collection process is divided into two phases on two time-lines; phase lines divide metrics collection (**Phase I**) from metrics validation (**Phase**

II); time-lines divide projects. At T_1 on **Time-line I**, a software project P_1 is measured and predefined metrics (**M**) are collected. At some point in time (T_2) on the same time-line, quality factor values (**F**) are collected from operational and test data on project P_1. At this point, **F** and **M** are tested against some validation criteria to determine if a suitable association exits between them. In essence, the objective is to see if **M** is in some way related to **F**. With **M** validated, a new project P_2 is entered into on **Time-Line II**. Once again, at **T1** metrics **M'** are collected. Note that **M'** is the same metric as **M** only with new and different values. This time, **M'** is used to **assess**, **control** or **predict** the quality of the P_2 as it is being developed. Again, at some point in time (T_2), quality factor values **F'** are collected. This time **M**, **M'**, **F** and **F'** are all subjected to a validation process in order to reevaluate the usefulness of the original metrics (**M**). (Schneidewind, 1992)

The following sections provide an example of this process being applied to a real-life system.

C. CASE STUDY

1. INTRODUCTION

The basis for this study is research conducted by Norman F. Schneidewind on the use of metrics on Space Shuttle software (Schneidewind, 1994). The purpose of this research

was to show that it is possible to collect and validate metrics which can be applied to future software projects to predict and control quality. In particular it proves that it is possible to statistically demonstrate an association between metrics and quality factors in order to support the premise that quality can be controlled in design by confining software metrics to certain critical value parameters.

The objective of the study is to find some relationship between metrics and quality factors that will enable the software quality manager to predict the quality of large-scale projects. In particular, it seeks to develop two types of design quality management tools; **Boolean Discriminator Functions** to control software quality and **Regression Equations** to predict future discrepancy report counts (Schneidewind, 1994). The methodology for meeting this objective is discussed below.

2. METHODOLOGY

The first step in this process is the application of metrics to a project to support quality functions as outlined by IEEE (IEEE, 1993) and Schneidewind (Schneidewind, 1992). Next, it is necessary to try and establish some relationship between a quality factor and one or more selected metrics. The identification of such a relationship is critical in order to develop the discriminator values and functions needed to control software quality. The final step is the application

of non-parametric statistical techniques to candidate metrics in order to identify those values that best support quality control. (Schneidewind, 1994)

Before continuing this discussion, two matters of importance need to be mentioned. First, in his approach to this study, Schneidewind found that data smoothing was required in order to rationalize the mass of data. Secondly, through follow-on research, Schneidewind was able to develop regression equations suitable for supporting quality prediction.

The foundation of this research lies in the application of two models: a ***Discriminative Power Model*** to identify quality control metrics; and a ***Prediction Model*** to identify quality prediction metrics. Each of these models and their application is discussed in the following sections.

D. DISCRIMINATIVE POWER MODEL

1. INTRODUCTION

As defined in the objective statement, the purpose of this study is to determine if sufficient relations exist between select metrics and a quality factor to enable the Software Quality Manager (SQM) to use these metrics to predict quality during the developmental phase of future software projects. The idea is to provide the SQM with some tool for managing quality control. The pre-requisite for the

application of this model is that sufficient software metric and quality factor data has been collected to provide a representative sample upon which statistical analysis can be performed. Ideally, data from a large number of software modules is desirable; in this case, data from 1489 modules used for Space Shuttle flight control was available and used.

The intended benefit of applying this Discriminative Power Model is the identification of one or more metrics with associated quality criteria for use by the SQM (Schneidewind, 1994). The following section describes the Discriminative Power Model, and the discriminative power identification and validation process.

2. MODEL PURPOSE

The Discriminative Power Model is used to determine if sufficient relationship exists between some metric **M** and some quality factor **F** to allow the SQM to apply **M** to future software development projects. As Schneidewind points out, a metric's validation process validates a metric with respect to one of the previously defined validity criteria (e.g., association, consistency, etc...) where each of the validity criteria support one or more of the three quality functions: quality control, quality assessment and quality prediction (Schneidewind, 1992). Therefore, application of this model is desirable in order to identify some critical metric value M_c and critical quality factor value F_c such that M_c can be

used to discriminate between modules that are above or below some F_C threshold. (Schneidewind, 1994)

For purposes of this study, it is desirable to identify some M_C that the SQM can use as an indirect measure of F_C when F_C is not available. For instance, consider a RSC from which metric M is collected. M_C can be used to evaluate that RSC in order to establish its potentially good ($M \leq M_C$) or potentially bad ($M > M_C$) quality.

3. MODEL DEFINITION

The principle tool used to validate M_C with respect to F_C in the Discriminative Power Model is *Contingency Table Analysis* and the *chi-square* (χ^2) criterion outlined by Conover (Conover, 1971). Other validity criteria include module misclassification, required inspections and product quality. (Schneidewind, 1992)

Table 7.1 illustrates a typical contingency table and will be used to aid further discussion of Contingency Table Analysis as part of the Discriminative Power Model.

TABLE 7.1 CONTINGENCY TABLE

	$M \leq M_C$	$M > M_C$
$F \leq F_C$	C_{11}	C_{12} Type II Misclassifications
$F > F_C$	C_{21} Type I Misclassifications	C_{22}

It is important to note that during this stage of metrics validation, all metric and quality factor data are available thus enabling the use of the contingency table. Therefore, using the table's criteria, all modules of interest can be classified in one of four categories based on whether their **M** value is $\leq M_c$ or $> M_c$ and their **F** value is $\leq F_{critical}$ or $> F_{critical}$. Note that M_c divides modules into two categories; those with $M > M_c$ are considered to be potentially poor in quality and should be examined; those with $M \leq M_c$ are considered acceptable. Metric **M** is then validated by demonstrating that it is able to divide the table such that C_{11} and C_{22} are relatively larger than C_{12} and C_{21} (Schneidewind, 1992).

For the perfect discriminator (M_c), $C_{12} = C_{21} = 0$. However, perfect discriminative metrics are seldom found; thus, other statistical methods such as **Chi-Square Contingency Table** are used to determine to what degree M_c serves as a perfect discriminator (Schneidewind, 1992). Other uses of the contingency table for metrics validation are explained below.

a. MISCLASSIFICATION

Two indicative measures of a metric's discriminative ability are the number of **Type I** and **Type II Misclassifications** it allows to occur (Schneidewind, 1992). In the Table 7.1, a Type I Misclassification occurs when a module containing more than a desired number of errors is improperly categorized as acceptable. Conversely, a Type II

Misclassification occurs when a module containing an acceptable number of error is categorized as unacceptable. Thus two measures of a metrics discriminative potential that can be drawn from the table are defined as (Schneidewind, 1994):

n (number of modules)

$P_1 = C_{21} / n$ (Percentage of Type I misclassifications) (1)

$P_2 = C_{12} / n$ (Percentage of Type II misclassifications) (2)

$P_{12} = (C_{21}+C_{12}) / n$ (Percentage of Type I & II misclassifications) (3)

b. INSPECTION

Another estimate of the discriminative power of a metric **M** with respect to quality factor **F** is the proportion of modules inspected and the portion that is wasted inspected (Schneidewind, 1994). This is explained by again looking at the Table 7.1. Here, all modules with $M > M_c$ are subject to inspection. As discussed in the above section, a number of those modules are improperly classified and thus represent wasted inspection efforts. Therefore, these added measures are defined (Schneidewind, 1994):

$I = (C_{22} + C_{12}) / n$ (Percentage of modules inspected) (4)

$RI = C_{22} / C_{12}$ (Ratio of useful to wasted inspections) (5)

c. QUALITY

A final estimate of a metric's discriminative power is the proportion of remaining quality factor values (e.g.,

defect-report-count) in modules not inspected (Schneidewind, 1992). This measure is found by summing the **F** count for all modules not inspected and dividing it by the total beginning **F** count from all modules. Hence, these final measures are given (Schneidewind, 1994):

RF	(sum of F for modules not inspected)	
TF	(sum of F prior to inspection)	
RFP = RF / TF	(percentage of F left after inspection)	(6)
RFD = RF / n	(density of F left after inspection)	(7)
RMP = C21 / n	(percentage of modules after inspection with F>0)	(8)

Having identified a model for metric validation defined, the next section focuses on the validation process itself.

4. THE ANALYTICAL PROCESS

The analysis process discussed here is based on the actual research conducted by Schneidewind on Space Shuttle software. Schneidewind, in his paper (Schneidewind, 1994), identifies and defines the metrics and quality factor shown in Table D-1 in Appendix D. These metrics and factor data were collected from 1489 flight software modules.

The first step in this analysis process is the selection of candidate metrics to be tested against the validity criteria to determine their potential for discriminators and predictors of quality. Initial scatter diagrams and histograms often provide a general indication of correlations among data. However, in this case, neither tool

provided any conclusive results. Instead, ***Principle Components*** and ***Factor Analysis*** were used to support preliminary candidate metric selection. (Schneidewind, 1994)

Definitions and descriptions of both Principle Components and Factor Analysis procedures are outlined below.

a. PRINCIPLE COMPONENTS ANALYSIS

The objective in Principle Components Analysis is to identify a few weighted combinations of metrics that: are independent; account for the greatest variation in the metrics; and have a high correlation with the given quality factor (Schneidewind, 1994). Figures D-1 and D-2 in Appendix D show the results of applying component analysis to the 13 metrics and quality factor identified in Table D-1.

In analyzing Figure D-1, the notion is to identify components that have a high value with respect to one component line (e.g., *Component 1*) and a low value with respect to the other on the other (e.g., *Component 2*) (Schneidewind, 1994). In this case it appears that ***stmts*** and ***nodes*** have the highest values along the Component 1 line (.319 and .311 respectfully) and low values along the Component 2 line (.12 and -.228 respectfully). A similar analysis technique is applied to Figure D-2.

In Figure D-2, lines from the origin to a metric represent the metric's contribution to a principal component. Here again ***stmts*** and ***nodes*** have the high values along the

Component 1 line and relatively low values along the Component 2 line. One analytical feature of this graph is the fact that the angle between any two metric lines is inversely proportional to the correlation between them (Schneidewind, 1994). Application of this feature to Figure D-2 indicates an apparent strong correlation between ***stmts*** and ***drcount***.

Principle Components Analysis provides one tool for metric assessment; additionally, Factor Analysis was used to support Principle Components Analysis findings. Subsequent application of Factor Analysis confirmed ***stmts*** and ***nodes*** as suitable metrics for validation testing. (Schneidewind, 1994)

b. CONCLUSIONS

In the analysis process, 13 metrics and one quality factor were analyzed. Principle Component and Factor Analysis of these metrics suggested that ***stmts*** and ***nodes*** were the most suitable of the 13 metrics for potential validation. Additionally, ***edges***, ***maxpath*** and ***avepath*** appeared to be viable contenders; future validation efforts can be expanded to include them. (Schneidewind, 1994)

Based on the above conclusions, The metrics ***stmts*** and ***nodes*** along with the quality factor ***drcount*** will be the subjects of interest for the remainder of this study.

5. DISCRIMINATIVE POWER VALIDATION MODEL APPLICATION

The result of the initial search for candidate metrics concluded that ***stmts*** and ***nodes*** appeared to be the best choice for purposes of this study. Further analysis, which compared ***drcount*** to ***stmts*** and ***nodes*** using the graphical representations of histograms, revealed a clustering of data points for low values of both the metrics and quality factor indicating that, if the critical value of ***drcount*** is low, then the associated critical values of ***stmts*** and ***nodes*** will be low (Schneidewind, 1994). Figure D-3 in Appendix D illustrates the plotting of the unsmoothed data points of ***stmts*** verses ***drcount***; a similar plotting of data for ***nodes*** verses ***drcount*** yielded the same results. From the figure, it is evident that data smoothing was necessary to extract useful information from the data.

a. DATA SMOOTHING

As Figure D-3 illustrated, little association is evident between ***stmts*** and ***drcount*** before data smoothing occurs. In order to refine the data, 92 of the 1489 initial modules were removed from analysis due to the fact that they contained a zero ***stmts*** count[11]. Data smoothing was then performed on the remaining 1397 modules by dividing the modules into 12 statistical classes representing 97.7% of the modules. Table D-2 in Appendix D shows the range and standard

[11]These modules contained assembly code which is not counted as statements for this project.

deviation for each of the remaining modules. Figures D-4 and D-5 show the plots of ***avestmts*** and ***avenodes*** verses ***avedrcount*** after data smoothing is performed (Schneidewind, 1994).

At this point, the analytical process can begin. Here, using contingency table analysis and the equations (1) through (8), Table D-3 is produced where the values for $\mathbf{D_c}$, $\mathbf{S_c}$ and $\mathbf{N_c}$ are derived from the first three classes in Table D-2 using rounded-down average (***Ave***) values. For example, from row one, average ***drcount*** = 0, average ***nodes*** = 9 and average ***stmts*** = 8 when rounded down to achieve whole numbers. Additionally, note that where values appear for both $\mathbf{S_c}$ and $\mathbf{N_c}$, these two metrics are applied compositely using the *OR* function. With Table D-3 defined, the statistical validation of ***stmts*** and ***nodes*** can be carried out.

b. STATISTICAL VALIDATION

This step in the model focuses on validating the selected metrics statistically. Here, Chi-square Analysis is applied where a high chi-square value and corresponding low significance values are desirable. In Table D-3, $\chi^2_c \gg \chi^2_s$ and $\alpha_c \ll \alpha_s$ (χ^2_s = 10.83, α_s = .001 and α_c = 0 to five places) for all cases; this provides sufficient statistical validation for all functions in the table (Schneidewind, 1994). Note, that the function ($\mathbf{S_c}$ = 8 *OR* $\mathbf{N_c}$ = 9) produces the highest chi-square of any single or combined metric pair application.

In addition to statistical validation, validation by application is desirable for both ***stmts*** and ***nodes***. This process is outlined next.

c. METRIC APPLICATION

At this point, the metrics ***stmts*** and ***nodes*** have been statistically validated; therefore, Table D-3 can now be used by the SQM to support quality management decisions. For example, if the SQM were interested in high-quality, high-inspection requirements, then the choice of small values for S_c and N_c would be appropriate. Conversely, for lower quality and lower inspection requirements, larger values for both metrics would be needed. Additionally, by using S_c and N_c in combination using the *OR* function, better results are possible then if the two metrics are used singly; Figure D-6 illustrates the effectiveness of this combination. Therefore, by varying the values selected for S_c and N_c, the SQM gains flexibility in the selection and application of software inspection requirements. One final issue to address is the tradeoffs inherent in metrics choice and use.

d. CONCLUSIONS

As mentioned above, choice of critical metric values (M_c) provides the SQM with the ability to modulate both quality and inspection requirements. Often there exits a tradeoff between quality requirements and the cost of module inspections (Schneidewind, 1994). For example, if the cost

associated with inspections were no object, the SQM could inspect all modules with (**S** > **8** *OR* **N** > **9**) (Table D-3) resulting in 67.9% of all modules being inspected. Conversely, if budgetary constraints limited inspections to 50% of all modules and a **RFP** of 18.1% was acceptable, only modules with (**S** > **48** *OR* **N** > **21**) could be inspected. Finally, as discussed earlier, the process of metric selection and validation is an iterative one where no single metric is considered permanently valid. Rather, as more software development and operational data are collected, current metrics should be subjected once again to the re-validation process. The next section introduces a power model for metrics validation that focuses on the predictive, vice discriminative, abilities of a metric.

E. METRICS PREDICTABILITY MODEL

1. INTRODUCTION

The discriminative power validation process is useful in proving that select metrics have a strong correlation to some quality factor. Another desirable feature of metrics, with respect to software quality measurement, is the ability to predict software quality in the absence of quality factors (Schneidewind, 1992). For example, if the number of discrepancies in a software module is unknown, the SQM would

like to be able to use ***stmts*** or ***node*** counts to predict the expected discrepancies in that module.

While the discriminative property of metrics applies to the segregation of low from high quality software modules, the predictive property applies to the use of prediction to estimate a particular module's behavior relative to some unknown quality factor. The following sections discuss the process of validating metrics for use as quality predictors.

2. PREDICTABILITY CRITERIA

In order for a metric **M** to satisfy the predictability criteria it must meet the following condition (Schneidewind, 1992):

$$\left|\frac{F_{\alpha_{T2}} - F_{\rho_{T2}}}{F_{\alpha_{T2}}}\right| < \beta_\rho \qquad (9)$$

In essence, for some function **f(M)** using metric **M'** (remember that **M'** = **M**) collected at time **T1** (refer to Figure 7.1, Time-line II), **f(M)** must be able to predict the quality factor $F_{\rho T2}$ at time **T2** with an accuracy of β_ρ. Figure 7.2 provides a graphical depiction of the variance criteria for **f(M)**. As this figure illustrates, in the ideal situation, **f(M)** = F_α; however this is seldom the case. Instead, metric **M'** and function **f(M)** are acceptable if **f(M)** falls within the tolerance of $F_\rho \pm$. (Schneidewind, 1992)

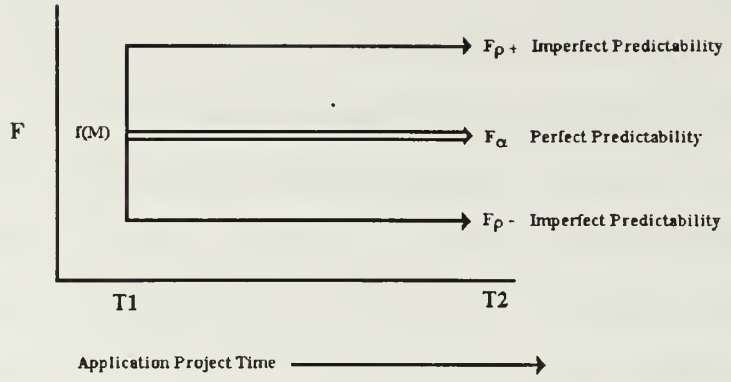

Figure 7.2 Predictability Criterion for f(M)
(Schneidewind, 1992)

The following sections continue this discussion by presenting the analytical portion of the Metrics Power Model.

3. STATISTICAL ANALYSIS

As a result of discriminative power modelling, ***stmts*** and ***nodes*** were identified as two potentially useful metrics. Figures D-4 and D-5 (Appendix D) show that ***avedrcount*** exhibits a linear relationship with respect to ***avenodes*** and non-linear relationship with respect to ***avestmts***. Additionally, Figure D-7 extends this discussion by illustrating that ***avedrcount*** exhibits a non-linear relationship with respect to the combination ***avenodes*** and ***avestmts***. These relationships provide the basis for regression analysis.

4. REGRESSION MODEL DEVELOPMENT

Based on the relationships between ***drcount*** and ***avenodes*** and ***avestmts*** outlined above, regression analysis was performed on the metrics ***stmts*** and ***nodes*** using the data from the 12 classes in Table D-2 to derive the following regression models (Schneidewind, 1994):

$$D_a(s) = \exp(.242+.00523S_a) \quad (10)$$

$$D_a(n) = -.262+.0658N_a \quad (11)$$

$$D_a(sn) = \exp(.348+.00194S_a+.00826N_a) \quad (12)$$

using the following notations:

S_a: *avestmts* used to produce $D_a(s)$ and $D_a(sn)$ or given value in $D_a(s)$ and $D_a(sn)$ used as predictors

N_a: *avenodes* used to produce $D_a(n)$ and $D_a(sn)$ or given value in $D_a(n)$ and $D_a(sn)$ used as predictors

d_a: *avedrcount* used to produce $D_a(s)$, $D_a(n)$, and $D_a(sn)$

$D_a(s)$: predicted *avedrcount* as a function of *avestmts*

$D_a(n)$: predicted *avedrcount* as a function of *avenodes*

$D_a(sn)$: predicted *avedrcount* as a function of *avestmts / avenodes*

D_a': actual *avedrcount*

Figures D-8, D-9 and D-10 plot the regression analysis equations (10), (11) and (12) against actual data from the modules. Although these plots demonstrate a fairly good fit between the actual and predicted values, further statistical validation is desirable.

5. PREDICTABILITY VALIDATION MODEL APPLICATION

At this point, the predictability validation model is applied in order to validate the degree to which the selected metrics can predict quality. Using the following equations (Schneidewind, 1994):

$$MRE = \frac{\sum_{k=1}^{C} \frac{|d_{ak} - D_{ak}|}{d_{ak}}}{C} \qquad C = \text{Number of Classes} \qquad (13)$$

$$MR = \frac{\sum_{k=1}^{C} (d_{ak} - D_{ak})}{C} \qquad (14)$$

$$MR = \frac{\sum_{k=1}^{C} (d_{ak} - D_{ak})}{C} \qquad (15)$$

Table 7.2 is produced:

TABLE 7.2 PREDICTABILITY VALIDITY CRITERION
(Schneidewind, 1994)

	MRE	MRE SD	MSE	MR	MR SD
$D_a(s)$	.247	.301	1.104	.0151	1.097
$D_a(n)$	.127	.117	.281	-.0000768	.554
$D_a(sn)$	.192	.388	.198	-.0300	.463

MRE: Mean Relative Error
MRE SD: MRE Standard Deviation
MSE: Mean Square Error
MR: Mean Residual
MR SD: MR Standard Deviation

Three evaluators of the goodness of fit of $D_a(s)$, $D_a(n)$ and $D_a(sn)$ with respect to the actual data (shown in Figures D-8 - D-10) are: **MRE**, **MSE**, and **MR** (Schneidewind, 1994). Here, **MRE** measures prediction error relative to ***avedrcount***. **MSE** helps by minimizing the sum of the variance and square of the bias of ***avedrcount***. Finally, **MR** provides a measure of the observed verses the predicted values of ***drcount***, without consideration for sign.

In analysis, residual plots are useful for demonstrating whether or not there is stability in predictions (Schneidewind, 1994). An examination of Table 7.2 reveals that there is no clear winner in all categories. For that reason, and because:

- Several predictors are more desirable than one.
- Often ***stmts*** is the only metric available early on in the software development cycle.
- It is desirable to revalidate all predictors using other, new data.

the predictor functions $D_a(s)$, $D_a(n)$, and $D_a(sn)$ are all considered useful and can be used in predicting software quality (Schneidewind, 1994). The next section describes the application of these functions to software projects.

6. PREDICTOR METRIC APPLICATION

Subsequent to the selection of predictor functions is the re-application of these functions to the project from

which they were derived in order to measure their validity. By obtaining random samples from the 1387 modules of upper and lower limits of ***stmst*** and ***nodes*** for three cases and computing the predicted and actual ***drcount***, Schneidewind was able to construct Table 7.3:

Table 7.3 SAMPLE OF PREDICTIVE METRICS APPLICATION
(Schneidewind, 1994)

S_a	N_a	D_a'	$D_a(s)$	$D_a(n)$	$D_a(sn)$
264.39	142.17	5.87	**5.09**	9.09	7.66
312.24	132.62	7.32	6.52	8.46	**7.76**
167.08	83.88	3.00	**3.05**	5.26	3.92

From this Table it is apparent that $D_a(n)$ failed in all three cases as a good predictor of actual ***drcount***. On the other hand, $D_a(s)$ and the metrics combination $D_a(sn)$ demonstrate favorable predictive abilities. While many more such tests are necessary before the results can be conclusive, this example illustrates the value and variability of predictor metrics in actual application.

In concluding this discussion it is noted that validated predictor metrics are suitable in three applications by the SQM (Schneidewind, 1994):

- When metrics for software modules are available and it is desired to form some prediction of the effect of those metrics on the module's quality.
- When metrics are available and it is desired to predict the effect of changes in design on the module's quality.
- When it is desired to use predictor metrics in the actual design process before coding begins.

As with discriminative metrics, predictor metrics provide the SQM with yet another tool by which quality analysis and prediction can be performed on software modules in the absence of any quantitative quality factor data.

F. CONCLUSIONS

Based on the research in this case study, the use of metrics to predict quality was successful (Schneidewind, 1994). It was found that Boolean *OR* functions could be developed for metrics to serve as discriminators of quality. It was also found that two metrics, when used together, might be better discriminators of quality than one metric alone. Finally, it was shown that regression equations can be developed which can serve as predictors of quality.

Of significant importance is the role that data smoothing played in this research; without data smoothing, these results would not have been achieved. While further research is necessary to continually improve and validate the process of metrics use, this study illustrates a starting point for such an effort.

CHAPTER VIII. CONCLUSIONS

A. THE COST OF QUALITY IMPROVEMENT

One argument with respect to improving software quality in the reuse library is that of its associated costs. In general, the DoD executive expects a significant return in terms of time or cost savings for resources dedicated to software quality improvement.

The response to this argument lies in the examination of the purpose of a quality measurement program. A quality measurement program provides the SQM with a resource management tool. For example, a SQM can apply select[12] predictor metrics to software components in order to distinguish the potentially good from bad. Cost savings are realized in this case when the SQM is able to minimize the wasted testing and inspection of good components and instead redirect critical resources to only those potentially bad components.

The major costs associated with the proposed quality improvement program lies in the establishment of a reliability database to support metrics evaluation. In many organizations (e.g., NASA), developing such a database involves significant

---

[12]Earlier discussions in this thesis outlined the process of metrics selection and validation.

time and money to test and document numerous software components. In this case, the reuse library has a particular advantage; it can rely on its users to test library components as part of their systems development. The reuse library need only then develop and maintain a record of user feedback on these testing results. Notably however, the success of this effort depends on willing cooperation and reliable feedback from users.

In any case, quality gains will not come without costs. Executive level managers should consider the long term benefits of investing in software quality improvement now.

B. CHOICE OF METRICS

Defect-report-count and *number-of-statements* were used as examples of metrics in this thesis; in practice, any number of metrics can be used. In fact, it is particularly desirable to start a quality measurement program with a number of candidate predictor metrics. Then, during the metrics validation process (Chapter VI), those metrics found unsuitable for quality prediction purposes can be eliminated.

As mentioned, Appendix A provides 15 sample metrics generally used in industry today. Additionally, IEEE (IEEE, 1992) lists a number of both direct and indirect metrics and includes a meaningful example (Annex C) illustrating the entire measurement methodology. Finally, Chapter VII provides a case study that outlines the process using the original

metrics that NASA selected for its Space Shuttle Software improvement effort.

Ultimately it is up to the organization to select and validate metrics suitable for their needs. Likewise, it is up to the organization to determine the critical value thresholds to which these metrics will be held. For example, a SQM may determine that a ***defect-report-count*** threshold of two is acceptable for information system type RSCs. On the other hand, the SQM may set a threshold value of zero ***defect-report-counts*** for flight critical type RSCs.

C. METRICS AS "THE" QUALITY IMPROVEMENT SOLUTION

As stated earlier in this thesis, a metrics program is not the solution to the DoD's software development problems. Rather, a good metrics program can play a supporting role in building a library of quality reusable components. To support DoD systems development, the reuse library begins by identifying the types of RSCs that its users need. Next, it collects RSCs of this type with the goal of developing them to a Level 4 status that is most valuable to the user. Finally, the SQM can focus on improving the quality of library assets. To that extent, metrics will play an important roll by providing the SQM with a tool to distinguish quality among components and to aid in determining the allocation of resources for RSC quality improvement.

D. SUMMARY

The author of this thesis does not suggest that the quality improvement methodology presented here is the only solution; rather, this is one methodology that can be used to support quality improvement. The author does believe that quality improvement measures are an important element in the development of a repository of high quality software components that can be used by systems developers to reduce their development time and costs.

The methodology presented here has been shown to work and can be readily applied in the reuse library environment. While acknowledging the costs associated with quality improvement, there is perhaps a greater cost associated with ignoring it. A real example of such danger lies in the history of software development methodologies.

In its infancy, software was often developed in an ad-hoc fashion without any formal methodology. Today, many years later, the software industry is still suffering the consequences for not having the foresight to develop and implement standard software engineering practices in the early days of software development. DoD today, starts a new era with the development of software reuse libraries. Component quality needs to be given proper attention now while the reuse initiative is in its infancy and provides a suitable ground-level entry point for any quality improvement effort; later

on, after huge repositories of software have been built may be too late to achieve quality improvement.

APPENDIX A

"BEST OF CLASS" MEASURES

A. INTRODUCTION

Siefert conducted one and one-half years of research on industry's use of software measures. To collect his data, he polled 350 organizations world-wide to determine the extent of their use of 39 industry software measures defined by IEEE (IEEE, 1988). He concluded his work by identifying 15 "Best of Class" measures evaluated on the criteria of their importance as well as their frequency and ease of use. Along with statistical preference data, he gathered information on the ways in which these measures are used and the standards by which they are evaluated. (Siefert, 1989) The next section discusses his recommendation for applying the results of his study.

B. APPLICATION

While all 39 of the IEEE measures are of importance, Siefert points out that he omitted a number of measures for which he received no significant response from his top 15 list . He goes on to suggest that the remaining 15 measures, which form a normalized composite of his research, provide a reasonable, low-risk starting point from which an organization can begin to build its measurement program. Siefert supports the methodology outlined in this thesis for establishing a software quality measurement program. In following this methodology, he suggests that an organization define its goals and then select and implement two or three of his "Best" measures using IEEE writings (IEEE, 1988) for guidance. He concludes by commenting that while his measures provide a statistical basis for metric selection, measures and standards selection should be driven by organizational experience and current technology. (Siefert, 1988)

The following tables are reproductions of the findings of Siefert with a few modifications in the interest of brevity and conserving space. The reader is directed to Siefert's work (Siefert, 1988) for a more detailed explanation of his research and findings.

TABLE A-1 "BEST OF CLASS" MEASURE'S USAGE

RANK	MEASURE	MEASURE USAGE
1	Fault Density	- Predict the remaining faults and system availability - Evaluate faults per N source lines of executable code - Applicable to predictive reliability models
2	Failure Rate	- Evaluate failure rates to operation time - Indicator of quality of software
3	Error Distribution	- Show correlation between module length and error distribution - Indicate need for further testing
4	Defect Density	- Measure reliability growth - Define defects per number of executable source LOC - Applicable to predictive reliability models
5	Cumulative Failure Profile	- Indicate software quality - Applicable to predictive reliability models - Supports measure # 11
5	Failure Analysis	- Measure software quality
6	Test Coverage	- Determine quality of testing - Evaluate test coverage adequacy
7	Fault Days Number	- Used for release decisions
8	Cyclomatic Complexity	- Used to estimate minimal cases - Show maintainability/testability - Determine complexity
9	Entries and Exits	(not available)
12	Functional Test Coverage	- Used for release decisions
11	Mean Time to Failure	- Indicator of quality of software - Calculated from slope extracted from graph of measure # 5
12	Halstead-Software Science Difficulty	- Determine latent defects content - Determine software size and complexity
13	Graph-Theoretic Complexity	- Used to determine where system level testing should concentrate
14	Source Listings and Documentation	- Used as part of inspection process
15	HW/SW Operational Availability	- Project systems availability

TABLE A-2 "BEST OF CLASS" MEASURE STANDARDS

RANK	MEASURE	STANDARD
1	Fault Density	- Raleigh distribution and historical data versus set goals - Faults counted per N LOC - Standards specific to project
2	Failure Rate	- Failure versus execution time - Project specific specifications - Better than past results
3	Error Distribution	- Normal distribution (desirable, but usually not found)
4	Defect Density	- Defects counted per N LOC - Less than .1 defect per 1,000 lines of code
5	Cumulative Failure Profile	- Parabolic shape and flattening over time
5	Failure Analysis	(not available)
6	Test Coverage	- Must exceed 80% - 100% of non-reused code tested
7	Fault Days Number	(not available)
8	Cyclomatic Complexity	- 10 or less (realistically, application dependent)
9	Entries and Exits	(not available)
10	Functional Test Coverage	- 100% of functions tested
11	Mean Time to Failure	- Better than past results - Minimum 2,000 hours (system requirements dependent)
12	Halstead-Software Science Difficulty	(not available)
13	Graph-Theoretic Complexity	- Comparison among other similar programs and past error history
14	Software Source Listings and Documentation	(not available)
15	Combined HW/SW Operational Availability	- Customer specifications and reliability growth function

APPENDIX B

SAMPLE SOFTWARE DEFECT REPORT

The attached pages serve as a sample software defect report for use in gathering discrepancy data on software reuse modules. The information contained in these reports was compiled from several sources. Keller (Keller, 1993), who manages and coordinates Shuttle software, provides insightful information on the defect data that IBM collects as part of its software quality management program. Florac (Florac, 1992), whose work with problem and defect counting at SEI, provides a general format as well as other items of interest in defect reporting. ANSI/AIAA (ANSI/AIAA, 1992), who have published a standard on practices for software reliability, contribute information on collecting discrepancy correction information.

As noted earlier, this report format and the information it contains serves as a starting point for discrepancy data collection. Although not comprehensive, it does include enough information to support the management of a software quality program.

SOFTWARE COMPONENT DEFECT REPORT

Date Observed: _____ Defect Report No.: _____
Organization's Name: _____
Point of Contact: _____

TYPE OF DEFECT			
SOFTWARE DEFECTS	X	OTHER DEFECTS	X
Requirements		Hardware	
Code		User Mistake	
Test Case		Operating System	
Design		Operations Mistake	
User Manual			

DEFECT FINDING ACTIVITY			
INTEGRATION OF:	X	FORMAL REVIEW OF:	X
Design		Plans	
Code		Requirements	
Test Procedure		Preliminary Design	
User Publications		Critical Design	
INSPECTIONS OF:		Test Readiness	
Requirements		Formal Qualification	
Preliminary Design		TESTING	
Detailed Design		Test Planning	
Code		Module Testing	
Operational Document		Component Testing	
Test Procedures		Integration and Testing	
CUSTOMER SUPPORT		Independent V & V	
Production/Deployment		Testing and Evaluation	
Installation		Acceptance Testing	
Operation		System Error Message	

SOFTWARE COMPONENT DEFECT REPORT

Description of the Defect/Problem:

DEFECT RELATED INFORMATION			
FINDING MODE	**X**	**RESOLUTION OF DEFECT**	**X**
Static (non-operational)		Fixed	
Dynamic (operational)		Waived with Workaround	
Unknown		Requirements Changed	
SEVERITY		Not a Problem	
SEVERE (must be fixed)		**DEFECT RELATED TO A PREVIOUS CHANGE**	
MAJOR (affects software performance)		Yes (Date:)	
MINOR (workaround available)		No	
INSIGNIFICANT (not visible to user)		Can't Tell	
TIME TO ISOLATE DEFECT		**CHANGE RECOMMENDATIONS**	
1 Hour or Less		Fault Correction	
1 Hour to 1 Day		Design Correction	
More than 1 Day		Clerical Correction	
Never Found		Specification Correction	
		Documentation Correction	

DEFECT DETECTION/PREVENTION INFORMATION

Identify which software lifecycle phase should have caught this defect and explain why it was not found.

○ Requirements Evaluation: _____

_____

_____

_____

_____

○ Design Inspection: _____

_____

_____

_____

_____

○ Code Inspection: _____

_____

_____

_____

_____

○ Development Testing: _____

_____

_____

_____

_____

○ Performance Testing: _____

_____

_____

_____

_____

○ Other: _____

_____

_____

_____

_____

SOFTWARE COMPONENT DEFECT REPORT

COMPONENT DATA	
Software Size (in LOC)	
Source Language Used	

COMPONENT FAILURE DATA (provide at least one)	
CPU Hours Since Last Failure	
Wall Clock Hours since Last Failure	
Number of Runs or Tests Since Last Failure	
Test Hours per Test Interval	
Number of Failures in Test Interval (above)	
Test Labor Hours Since Last Failure	

DEFECT CORRECTION DATA	
Date and Time Correction Made	
Labor Hours to Make Correction	
Provide one of the following:	
CPU Hours to Fix Defect	
Number of Runs to Effect Fix	
Wall Clock Hours to Effect Fix	

Miscellaneous Comments:

APPENDIX C

AMPLIFICATION OF THE SOFTWARE METRICS VALIDATION METHODOLOGY

A. INTRODUCTION

A predictive metric is considered valid, if and only if it is proven to possess a high degree of correlation with the quality factor it replaces. Additionally, a predictive metric may prove to be valid with respect to only a subset of the six metrics validation criterion; correlation, tracking, consistency, predictability, discriminative power and reliability. (Schneidewind, 1992)

Before starting this discussion, it is useful to establish an understanding of the rationale for metrics validation. Schneidewind points out that the purpose of software metrics validation is to prove the following:

$$\text{IF } R[M] \Leftrightarrow R[F] \text{ THEN } \{R[M] \Leftrightarrow R[F]\} \Rightarrow \{R[M'] \Rightarrow R[F']\}? \qquad (1)$$

Consider a project P_1 from which some metric (**M**) and quality factor (**F**)[1] have been collected. This relation then suggests that if some relation (**R**) between **F** and **M** on P_1 can be statistically validated with respect to some validity criteria, subject to a threshold value β and confidence level of α, then the **R** in P_1 should hold true in another project P_2[2]. Concisely stated, if **M** can be mapped to **F** on P_1 and validated then **M'** should map to **F'** on P_2. Hence, the essence of the validation process is to validate **M** with respect to one or more of the validity criterion using a threshold value β and a confidence level of α such that (1) holds true (Schneidewind, 1992).

The following examples illustrate metrics validation with respect to each of the six validity criteria. These examples are drawn from the publishings of Schneidewind (Schneidewind, 1992) and IEEE (IEEE,1992). Terms in parenthesis indicate an alternate choice of syntax for a specific validity criteria.

B. METRICS VALIDATION EXAMPLES

1. ASSOCIATION (CORRELATION)

Given R^2 (where **R** is the linear correlation efficient for

---

[1]**F** and **M** will be used hereon in place of the terms **Quality Factor** and **Metric** respectively.

[2]The reader is directed to Figure 7.1 in Chapter VII for an illustration of this methodology.

F and **M**) which represents the variation in **F** due to variations in **M** and β which represents some threshold value of R^2, the ***association*** test for a metric specifies that $R_2 > \beta$ must hold true a given confidence level α. This test seeks to show that sufficient linear correlation exits between **F** and **M** such that **M** can be used as a substitute for **F** when **F** is unknown. (Schneidewind, 1992)

For example, if **R = 0.7** for metric ***number-of-statements*** and quality factor ***defect-report-count*** taken from a sample software component, then $R^2 = .49$ which suggests that 49 percent of the variation in the number of defect reports is explained by the number of statements in this module. If β were chosen to be equal to 0.7, or greater, then metric ***number-of-statements*** would fail the validity test with respect to the association validity criteria.

2. TRACKING

Metric validation with respect to ***tracking*** is used to determine the ability of a **M** to change in unison with **F** for a given component over a period of time. Schneidewind describes this relationship for some component C_1 with associated metric M_1 and quality factor F_1 using the following notations:

$$\begin{aligned} &M_1(T_1) > M_1(T_2) \Leftrightarrow F_1(T_1) > F_1(T_2) \quad (\text{where } T_2 > T_1) \\ &M_1(T_1) = M_1(T_2) \Leftrightarrow F_1(T_1) = F_1(T_2) \\ &M_1(T_1) < M_1(T_2) \Leftrightarrow F_1(T_1) < F_1(T_2) \end{aligned} \qquad (2)$$

In essence, in order for a metric to be valid with respect to the ***tracking*** criteria, any change in **F** between times T_1 and T_2 must be accompanied with a proportional change in the same direction of **M**. If **M** can be proven to behave according to the properties outlined in (2), it can then be used as an indirect measure of **F** when **F** is unknown. (Schneidewind, 1992)

For example, consider project P^1 where ***number-of-statements*** and ***defect-report-count*** are given as $M^1 = 4000$ and $F_1 = 40$ at time T_1 and $M^2 = 2000$ and $F_1 = 20$ at time T_2. From this information it appears that **M** changes proportionally and in the same direction as **F**. If this relation between **F** and **M** is proven to hold over a representative sample of software components, then ***number-of-statements*** can be considered suitable for tracking ***defect-report-count*** over the project's lifecycle.

3. CONSISTENCY

The ***consistency*** validity test proves that the rank ordering of a set of metrics associated with a set of projects correlates to the rank ordering of the quality factors associated with the same set of projects (Schneidewind, 1992).

For example, consider projects P_1, P_2 and P_3 where ***number-of-statements*** for each project is given as $M^1 = 4000$, $M^2 = 2000$ and $M^3 = 1500$ respectively. Here, the rank ordering based on a low

value being most desirable is: $M^3 > M^2 > M^1$. In order for *number-of-statements* to be valid, the *defect-report-count* for the group of projects must demonstrate the same rank ordering (i.e., $F^3 > F^2 > F^1$).

Metrics are validated with respect to the *consistency* criteria by ensuring the rank correlation coefficient **r** between **F** and **M** exceeds a predefined threshold with a certain level of confidence. Specifically, the following must hold true: (Schneidewind, 1992)

$$r > \beta_{critical} \text{ with a given } \alpha_{critical} \qquad (3)$$

Consider the example where $\beta_{critical} = 0.6$, $\alpha_{critical} = .05$ and for *number-of-statements* and *defect-report-count*, $r = .7$ with $\alpha = .05$. Since $r > \beta_{critical}$ with an acceptable confidence level, if this 70% ranking for **F** and **M** is proven to exist over a representative sample of software components, then *number-of-statements* appears to be consistent with *defect-report-count* and can be used in ranking associated components in terms of quality.

4. PREDICTABILITY

In order for **M** to satisfy the predictability criteria it must satisfy the following condition:

$$\left| \frac{F_{\alpha_{T2}} - F_{\rho_{T2}}}{F_{\alpha_{T2}}} \right| < \beta_\rho \qquad (4)$$

In essence, for some function **f(M)** using metric **M'** collected at time **T1**, **f(M)** must be able to predict the quality factor $F_{\rho_{T2}}$ at time **T2** with an accuracy of β_ρ. (Schneidewind, 1992)

Consider project P_1 where at T_1 *number-of-statements*, given as M' = 5000, predicts *defect-report-count* $F_{\rho_{T2}} = 30$ where project standards require $\beta_\rho = .20$. In order for **M'** to be valid with respect to the predictability criteria, $F_{\alpha_{T2}}$ must be less than 50. If the application of **M'** using a representative sample of components shows that **M'** meets the requirements of (4), then *number-of-statements* can be considered a suitable predictor of *defect-report-count* and can be applied in the context of software quality control.

5. DISCRIMINATIVE POWER

To meet the *discriminative power* test, Schneidewind points out that a critical metric value M_c for a given critical quality factor value F_c must be able to classify metric M_1 from

component C_1 with a specified α such that:

$$M_1 > M_c \Leftrightarrow F_1 > F_c \quad \text{and}$$
$$M_1 \leq M_c \Leftrightarrow F_1 \leq F_c \qquad (5)$$

In short, M_c must be able to distinguish between high and low quality components with a given level of confidence. (Schneidewind, 1992)

IEEE suggests the use of the Mann-Whitney Test and Chi-square test (contingency table) for this type of metric validation (IEEE, 1992). Table C-1 will be used to illustrate the application of contingency table analysis to a project.

TABLE C-1 CONTINGENCY TABLE

$M_c = 4000$ $F_c = 40$	$M_1 \leq M_c$	$M_1 > M_c$
$F_1 \leq F_c$	$O_{11} = 3$	$O_{12} = 0$
$F_1 > F_c$	$O_{21} = 1$	$O_{22} = 4$

O_{ij} = count of observations in cell i,j

The values used in Table C-1 illustrate that for project P_1, one component ($O_{21} = 1$) is observed to have passed the acceptable quality test ($M_1 \leq M_c$) yet failed the qualify factor test $F_1 > F_c$. While a perfect M_c is difficult to find, the objective is to validate a metric with respect to the predictability criteria after (5) is proven to hold true over a representative sample of components. If this is the case then, **M** can serve as a discriminator of quality in various quality functions.

6. REPEATABILITY (RELIABILITY)

A metric passes the ***repeatability*** criteria if it demonstrates a given percentage rate of success when validated with respect to one or more of the validity criterion described above. Specifically, Schneidewind proposes the criteria that for some **M**, the following must hold true over a given set of validity criterion (Schneidewind, 1992):

$$N_{i\_\_} / N_i > \beta_{i\_\_} \qquad (6)$$

Here, $N_{i\_\_}$ represents the number of successful validations of **M**, N_i represents the total number of validity tests **M** is subjected to and

$\beta_{i\_\_}$ represents a threshold evaluation criteria. Thus, this relation states that the percentage of successful validations of a metric with respect to a given set of criteria must exceed a certain critical value in order to provide confidence in that metric's use in software quality functions.

C. CONCLUSIONS

The intent of this appendix is to provide the reader with an example of metrics validation with respect to each of the six validity criterion. Further information is available in the writings of Schneidewind (Schneidewind, 1992) and IEEE (IEEE, 1992). In particular, Schneidewind provides a good discussion on the purpose and use of metrics. In his paper he provides an useful table (Appendix A) which correlates the quality functions assessment, control and prediction to the six validity criterion and gives examples of the statistical methods by which metrics can be validated.

APPENDIX D

CHAPTER VII TABLES AND FIGURES

TABLE D-1 SHUTTLE SOFTWARE METRICS AND QUALITY FACTOR
(Schneidewind, 1993)

Metric	Metric Description
eta1	unique operator count
eta2	unique operand count
n1	total operator count
n2	total operand count
stmts	total statement count
loc	total non-commented lines of code
comments	total comment count
nodes	total node count (in control graph)
edges	total edge count (in control graph)
paths	total path count (in control graph)
cycles	total cycle count (in control graph)
maxpath	maximum path length (edges in control graph)
avepath	average path length (edges in control graph)
Quality Factor	**Quality Factor Description**
drcount	discrepancy reports covering discrepancies (defects) between planned and actual requirements, design, and code as obtained form inspection of the documentation and test of the code

TABLE D-2 SMOOTHED METRICS DATA FOR 12 CLASSES
(Schneidewind, 1993)

Class	*stmts*			*nodes*			*dr count*		
	Range	Ave	S.D.	Range	Ave	S.D.	Range	Ave	S.D.
1	1-34	8.97	9.18	3-288	9.86	20.89	0-31	.65	2.17
2	35-68	48.72	10.14	3-60	21.82	13.68	0-13	1.57	2.33
3	69-103	85.12	10.62	3-79	35.35	18.91	0-13	2.08	2.57
4	104-137	119.58	9.30	5-119	45.95	28.10	0-18	2.79	3.84
5	138-171	156.55	9.28	5-147	56.13	39.34	0-22	4.00	4.43
6	172-206	189.70	10.83	5-167	75.08	44.02	0-13	3.95	4.15
7	207-240	222.77	9.48	5-156	90.64	40.14	0-13	4.91	3.22
8	241-275	254.37	11.50	5-166	71.04	61.04	0-12	3.95	4.09
9	276-309	294.20	10.56	5-167	95.67	56.17	0-34	5.67	8.25
10	310-343	320.22	10.47	5-171	65.33	59.04	0-10	4.22	3.90
11	344-378	357.86	9.70	5-338	156.00	81.61	1-37	9.93	9.36
12	379-412	397.88	6.22	5-232	145.25	94.17	1-26	10.38	9.55

TABLE D-3 DISCRIMINATIVE POWER VALIDITY EVALUATION
(Schneidewind, 1993)

D_c	S_c	N_c	P_1	P_2	P_{12}	I	RI	RFP	RFD	RMP	χ^2_c
0	8	-	4.72	27.3	32.0	63.8	1.34	9.89	.183	4.72	258
0	-	9	12.6	17.8	30.4	46.4	1.61	28.6	.528	12.6	208
0	8	9	2.79	29.5	32.3	67.9	1.31	4.11	.0759	2.79	286
1	48	21	7.23	20.0	27.2	42.3	1.11	18.1	.335	12.0	261
2	85	35	6.73	16.8	23.5	31.3	.86	27.8	.513	18.2	237

D_c: Calculated critical value of *drcount*
S_c: Calculated critical value of *stmts*
N_c: Calculated critical value of *nodes*
χ^2_c: Calculated chi-square

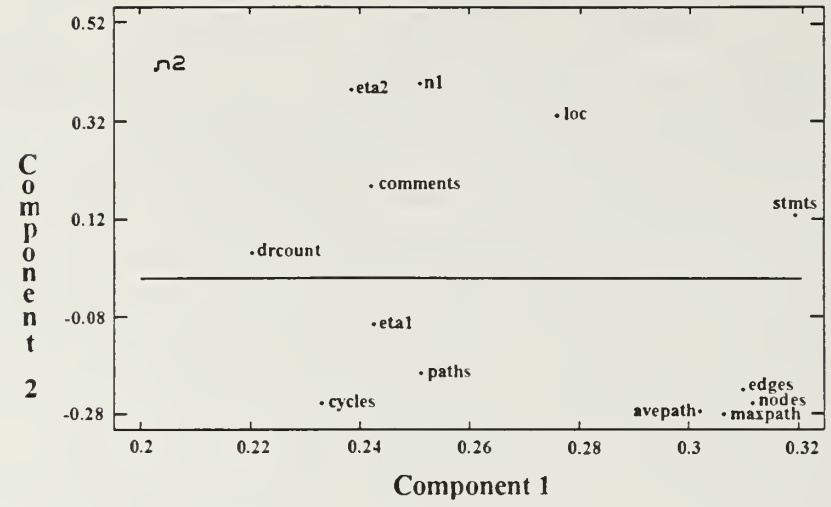

Figure D-1 Principle Components Weights (Schneidewind, 1993)

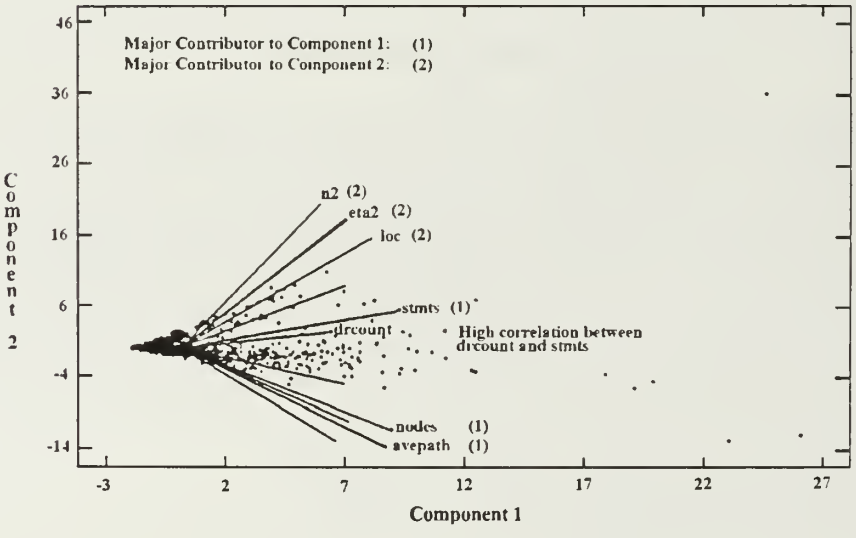

Figure D-2 Major Contributors to Components (Schneidewind, 1993)

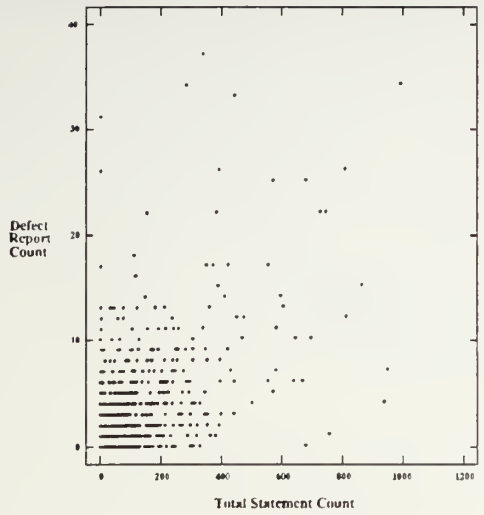

Figure D-3 Defects vs. Statements (Schneidewind, 1993)

Average Defect Count in Classes versus
Average Statement Count in Classes

Figure D-4 Average Defects vs. Average Statements
(Schneidewind, 1993)

Average Defect Count in Classes versus
Average Node Count in Classes

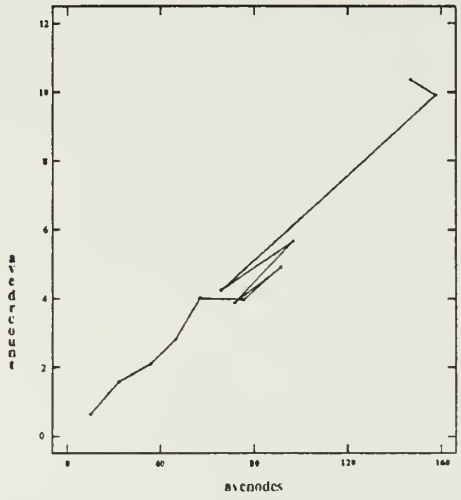

Figure D-5 Average Defects vs. Average Nodes
(Schneidewind, 1993)

Quality: Remaining drcount and Modules
with drcount > 0, after Inspection

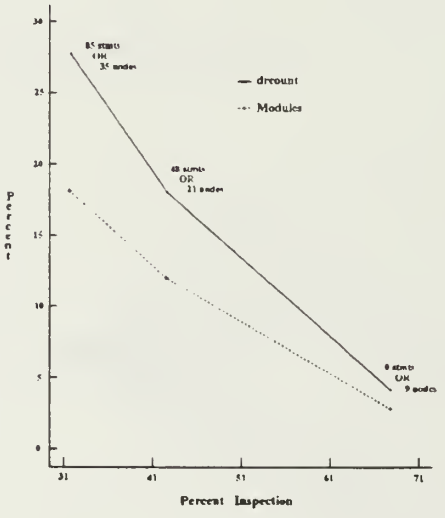

Figure D-6 Quality vs. Inspection (Schneidewind, 1993)

Average Defect Report Count versus Average Statement Count and Average Node Count

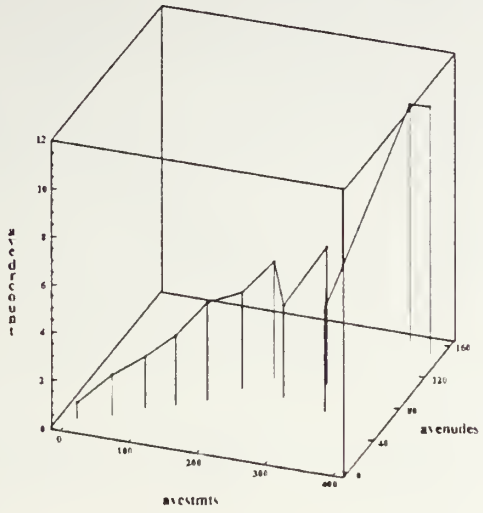

Figure D-7 Defects vs. Statements and Nodes (Schneidewind, 1993)

avedrcount = exp(.242 + (.00523 * avestmts))

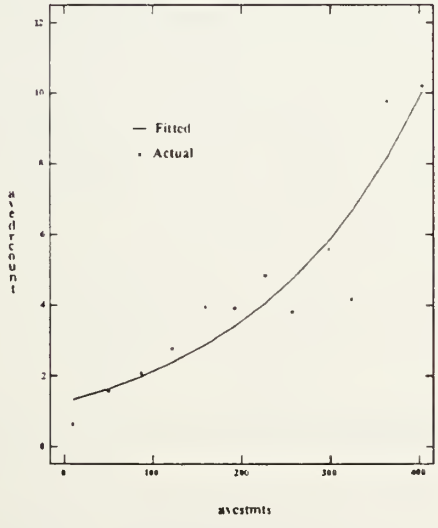

Figure D-8 Avedrcount vs. Avestmts (Schneidewind, 1993)

Average Defect Count = -.262 + (.0658 * Average Node Count)

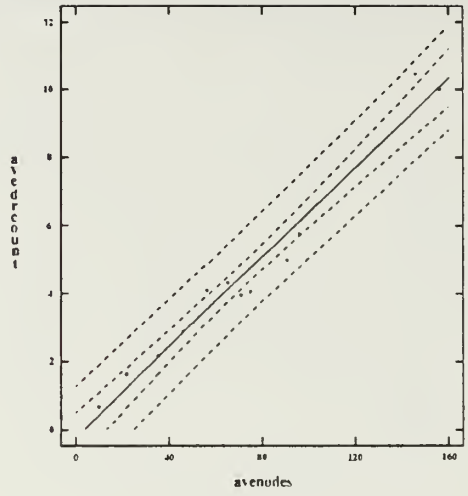

Figure D-9 Average Defects vs. Average Nodes (Schneidewind, 1993)

Avedrcount = exp(.348 + (.00194 * avestmts) + (.00826 * avenodes))

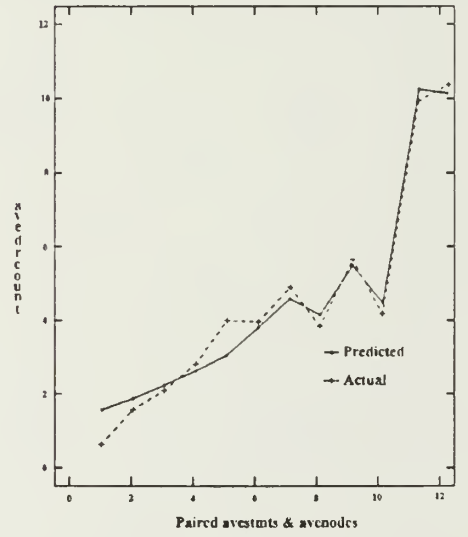

Figure D-10 Actual and Predicted Avedrcount (Schneidewind, 1993)

LIST OF REFERENCES

Basili, V. R., and Weiss D. M., "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 6, pp. 728-738, November 1984.

Booch, G., *Software Engineering with Ada*, The Benjamin/Cummings Publishing Company, 1987.

Conover, W. J., *Practical Nonparametric Statistics*, John Wiley & Sons, Inc., 1971.

"Dictionary of Measures to Produce Reliable Software," *IEEE Standard 982.1-1988.*

DISA/JIEO/CIM Software Reuse Program, *Software Reuse Metrics Plan*, Version 4.1, August 1993.

DISA/CIM Software Reuse Program, *Reusable Software Component Certification Guidelines for Ada Implementation Component Types*, Version 1.0, February 1993.

Dunn, R. and Ullman, R., *Quality Assurance for Computer Software*, McGraw-Hill Book Company, 1982.

DoD Reuse Executive Steering Committee, *DoD Software Reuse Vision and Strategy*, July 1992.

Foreman, J., "Software Technology for Adaptable Reliable Systems," talk presented at the Software Technology Conference, 5th, Salt Lake City, Utah, 21 April 1993.

"Guide to the Use of Dictionary of Measures to Produce Reliable Software," *IEEE Standard 982.2-1988.*

"IEEE Standard for a Software Quality Metrics Methodology," *IEEE Standard 1061-1992.*

"IEEE Standard Glossary of Software Engineering Terminology," *IEEE Standard 610.12-1990.*

Institute for Defense Analysis Report P-1191, *A Common Programming Language for the Department of Defense--Background and Technical Requirements*, by D. A. Fisher, pp. 2-3, June 1976.

Keller, T. W., "Shuttle Primary Avionics Software Reliability Approach," talk presented at the Software Technology Conference, 5th, Salt Lake City, Utah, 21 April 1993.

Merritt, S., "Managing Assets in a Software Reuse Library," talk presented at the Software Technology Conference, 5th, Salt Lake City, Utah, 21 April 1993.

Musa J. D., et al., *Software Reliability: Measurement, Prediction, and Application*, McGraw-Hill Book Company, 1987.

Ogush, M., "A Software Reuse Lexicon," *CrossTalk*, pp. 41-45, December 1992.

"Recommended Practice for Software Reliability," *ANSI/AIAA Standard R-013-1992*.

Reindollar, P. D., "The Establishment of a Metric Program," talk presented at the Software Technology Conference, 5th, Salt Lake City, Utah, 21 April 1993.

Schneidewind, N. F., "Controlling and Predicting the Quality of Space Shuttle Software Using Metrics," paper presented at the Proceedings of the 1994 Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon, April 10-12, 1994.

Schneidewind, N. F., "Methodology for Validating Software Metrics," *IEEE Transactions on Software Engineering*, Vol. 18, No. 5, pp. 410-422, May 1992.

Schneidewind, N. F., "New Software-Quality Metrics Methodology Standard Fills Measurement Need," *Computer*, pp. 105-106, April 1993.

Schneidewind, N. F. and Keller T. W., "Applying Reliability Models to the Space Shuttle," *IEEE Software*, pp. 28-33, July 1992.

Sheldon, F. T., et al., "Reliability Measurement: from Theory to Practice," *IEEE Software*, pp. 13-20, July 1992.

Siefert, D. M., "Implementing Software Reliability Measures," *The NCR Journal*, Vol. 3, No. 1., pp.24-34, March 1989.

Software Engineering Institute Technical Report CMU/SEI-92-TR-22, *Software Quality Measurement: A Framework for Counting Problems and Defects*, by W. A. Florac, September 1992.

Sommerville, I., *Software Engineering*, Addison-Wesley Publishing Company, 1992.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria VA 22304-6145	2
2.	Library, Code 052 Naval Postgraduate School Monterey CA 93943-5002	2
3.	Commandant of the Marine Corps Code TE 06 Headquarters, U.S. Marine Corps Washington DC 20380-0001	1
4.	Systems Management Department Naval Postgraduate School Attn: Professor N.F. Schneidewind Monterey CA 93940-5000	1
5.	Systems Management Department Naval Postgraduate School Attn: Cmdr W.B. Short Monterey CA 93940-5000	1
6.	Defense Information Systems Agency Software Reuse Program Management Office Code: CIM/XR 500 N. Washington Street Suite 200 Falls Church VA 22046	1
7.	Ms. Joan McGarity Software & Data Management Division Naval Computer & Telecommunications Command NCTC 4401 Massachusetts Avenue, N.W. Washington DC 20394-5000	1
12.	Mrs. Elfriede Warburton 2597 North Brooke Road Fort Meade FL 33841	1
13.	Capt Kenneth M. Warburton 395 Del Monte Center # 126 Monterey CA 93940	1